НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО »
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2020р.

Дипломний проект

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерні системі та мережі»
спеціальності 123 «Комп'ютерна інженерія»
на тему: «Розробка клієнт-серверної системи трансляції програмного коду між графічним та текстовим представленням»

Виконав:
Студент IV курсу, групи ІО-62,
Замалдінов Олексій Олегович _____

Керівник:
Ст. викладач,
Алещенко Олексій Вадимович _____

Консультант з нормоконтролю
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Ас. кафедри СПіСКС ФПМ,
Радченко Костянтин Олександрович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ - 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти - перший (бакалаврський)
Спеціальність - 123 “Комп’ютерна інженерія”
Освітньо-професійна програма - “Комп’ютерні системі та мережі”

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2020р.

ЗАВДАННЯ
на дипломний проект студенту
Замалдінову Олексію Олеговичу

1. Тема проекту (роботи) “Розробка клієнт-серверної системи трансляції програмного коду між графічним та текстовим” керівник проекту Алещенко Олексій Вадимович, ст. викладач, затверджені наказом по університету від “07” травня 2020 року №1081-с
2. Термін подання студентом проекту _____
3. Вихідні дані до проекту
   - Мова програмування PHP
   - Бібліотека GoJS
   - Мова розмітки HTML та каскадні стилі CSS
4. Зміст роботи
   - Аналіз існуючих графічних редакторів
   - Способи опису програмних алгоритмів
   - Розробка програмного продукту
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)
   - Структура та зв’язок між вузлами розробленої системи
   - Взаємодія модулів програми
   - Алгоритм обробки запитів

6. Консультанти розділів проекту

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	проф. Сімоненко В. П.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітки
1.	Затвердження теми проекту	01.09.2019	
2.	Вивчення та аналіз завдання	18.01.2020	
3.	Аналіз існуючих рішень	04.02.2020	
4.	Вибір засобів розробки	11.02.2020	
5.	Розробка програмного продукту	08.03.2020	
6.	Тестування	10.04.2020	
7.	Оформлення пояснювальної записки	01.05.2020	
8.	Передзахист	26.05.2020	
9.	Захист	18.06.2020	

Студент Олексій ЗАМАЛДІНОВ

Керівник роботи Олексій АЛЕЩЕНКО

Анотація

В бакалаврській дипломній роботі розглядається процес генерації графічних схем алгоритмів за програмними кодами. Як практична сторона реалізована система трансляції програмного коду в проміжне представлення, яке призначене для подальшої генерації блок-схем алгоритмів. Програма дозволяє генерувати графічні схеми, зберігати та відкривати їх. Програмний продукт був створений на мовах HTML, CSS, JavaScript, PHP у інтегрованому середовищі розробки Visual Studio Code. Для візуалізації, та взаємодії використовується браузер з підтримкою технологій HTML5, CSS3, JS (ECMAscript 6).

Загальний обсяг дипломного проекту: опис альбому (1 арк.), технічне завдання (3 арк.), пояснювальна записка (52 арк.), додатки (12 арк.).

Ключовы слова: блок-схема, алгоритм, інформаційні технології, веб-сервіс, PHP, GoJS, HTML, CSS, JS.

Annotation

In the bachelor's thesis the process of generating graphical schemes of algorithms by program codes is considered. As a practical side, the system of translation of program code into an intermediate representation is implemented, which is intended for further generation of block diagrams of algorithms. The program allows you to generate graphics, save and open them. The software product was created in HTML, CSS, JavaScript, PHP in the integrated development environment Visual Studio Code. A browser with support for HTML5, CSS3, JS (ECMAscript 6) technologies is used for visualization and interaction.

The total volume of the diploma project: album description (1 sheet), terms of reference (3 sheets), explanatory note (52 sheets), appendices (12 sheets).

Keywords: block diagram, algorithm, information technology, web service, PHP, GoJS, HTML, CSS, JS.

Опис альбому

№ з/п	Формат	Позначення	Найменування	Кількість аркушів	Примітка
			Документація загальна		
			Заново розроблена		
1	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	3	
2	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	53	
3	А3	ІАЛЦ.467100.004 Д1	Структурна схема	1	
4	А3	ІАЛЦ.467100.005 Д2	Функціональна схема	1	
5	А3	ІАЛЦ.467100.006 Д3	Принципова схема	1	
6	А4	ІАЛЦ.467100.007 Д4	Ключові елементи коду програми	9	

					ІАЛЦ.467100.001 ОА		
Змн	*Літ.*	*№ докум.*	*Підпис*	*Дат*			
Розроб.		*Замалдінов О. О.*			*Розробка клієнт-серверної системи трансляції програмного коду між графічним та текстовим представленням* *Опис альбому*	*Літ.*	*Арк.* *Аркушів*
Перевір.		*Алещенко О. В.*					*1* *1*
Т. Контр						*НТУУ “КПІ”, ФІОТ, гр. ІО-62*	
Н. Контр		*Сімоненко В. П.*					
Затверд.		*Стіренко С. Г.*					

Технічне завдання

Зміст

1. Найменування та область застосування

Дане технічне завдання розповсюджується на розробку клієнт-серверної системи транслювання коду з текстового вигляду в графічний. Область застосування: навчання основам програмування більшості мов програмування за допомогою графічного представлення алгоритмів.

2. Підстава для розробки

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп’ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. І. Сікорського».

3. Мета та призначення роботи

Метою даного проекту є розробка системи генерації графічних схем алгоритмів по коду.

4. Джерела розробки

Джерелом розробки є науково-технічна література з теорії і практиці структурного програмування, публікації в періодичних виданнях, довідники, публікації в Інтернеті з даних питань.

5. Технічні вимоги

5.1. Вимоги до продукту розробки

- Відносно простий і інтуїтивно-зрозумілий інтерфейс системи.
- Можливість збереження і відновлення створених раніше графічних схем.
- Незалежність - система повинна бути автономна і не залежати від платформи (операційної системи) та браузеру.
- Технічна коректність і актуальність інформації, що становить наповнення системи.

5.2. Вимоги до програмного забезпечення

- Браузер з підтримкою HTML5 та CSS3 (клієнт)
- PHP версії 7.3 або вище (сервер)

5.3. Вимоги до апаратної частини

- Комп’ютер на базі процессора Intel Pentium II 233 або вище.
- Оперативна пам’ять не менше 128 Мбайт.
- Підключення до глобальної мережі Інтернет

6. Етапи розробки

Етап	Дата
Вивчення та аналіз завдання	18.01.2020
Аналіз існуючих рішень	04.02.2020
Вибір засобів розробки	11.02.2020
Розробка програмного продукту	08.03.2020
Тестування	23.04.2020
Оформлення пояснювальної записки	01.05.2020

Пояснювальна записка

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ООП	Об'єктно-орієнтоване програмування
TCP	(англ.) Transmission Control Protocol - Протокол керування передачею
IP	(англ.) Internet Protocol - Міжмережевий протокол
ОС	Операційна система
HTTP	(англ.) Hyper Text Transfer Protocol - протокол передачі гіпер-текстових документів
XHTML	(англ.) Extensible Hypertext Markup Language - Розширювана мова розмітки гіпертексту
HTML5	(англ.) HyperText Markup Language, version 5 - Мова розмітки гіпертексту 5ої версії
CSS3	(англ.) Cascading Style Sheets, version 3 - Каскадні таблиці стилів 3ої версії
JS	JavaScript
PHP	(англ.) PHP: Hypertext Preprocessor - PHP: гіпертекстовий препроцесор
ПК	Персональний комп’ютер
IT	Інформаційні технології
API	(англ.) Application Programming Interface - Прикладний програмний інтерфейс
FTP	(англ.) File Transfer Protocol - Мережевий протокол передачі файлів
DOM	(англ.) Document Object Model - Об’єктна модель документа
XML	(англ.) Extensible Markup Language - Розширювана мова розмітки
UI	(англ.) User Interface - Інтерфейс користувача

URL	(англ.) Uniform Resource Locator - єдиний вказівник на ресурс
UML	(англ.) Unified Modeling Language - уніфікована мова моделювання
ГОСТ	(рос.) Государственный стандарт
ISO	(англ.) International Organization for Standardization - Міжнародна організація зі стандартизації

ВСТУП

На даний момент, інформаційні технології являють собою одну з найбільш цікавих та затребуваних галузей, що глибоко проникає в усі можливі сфери нашого життя та щоденно перетинається як з функціонуванням світових корпорацій, так і відіграє важливу роль в буденних справах звичайного користувача. До того ж, інформаційні технології - це відносно нова галузь, тому вона ще знаходиться на етапі безперервного стрімкого розвитку, який має бути доволі тривалим.

Професії, пов’язані з інформаційними технологіями вважаються дуже затребуваними та високооплачуваними, чим обумовлена зростаюча популярність галузі та велика кількість людей, що кожного дня вирішує приєднатися і починає свій шлях у світі кодів та розробок. Наприклад, відповідно до статистики 2018 року, кількість вже існуючих членів IT-суспільства, поповнилася на 2,5 млн нових програмістів [1].

Особливістю сфери інформаційних технологій є велика різноманітність, тому у залежності від конкретного відгалуження, також варіюється поріг входження. Доволі часто у нових розробників виникають складнощі в розумінні роботи алгоритмів. Для вирішення цієї проблеми найбільш доступним способом, доцільним є використання програмного забезпечення, що формуватиме візуальне відображення роботи алгоритмів.

Графічний спосіб представлення алгоритмів, тобто їх візуальне відображення, відбувається за допомогою використання спеціальних графічних засобів, що називаються блок-схемами. Зазвичай, блок-схеми використовують у роботі з послідовними функціональними підходами. За використання об’єктно-орієнтованого підходу, блок-схеми не здатні повною мірою відобразити функціонал алгоритму.

Більшість програмістів починають крокувати інформаційними

технологіями саме через ознайомлення з основами ООП. Одним з доволі важких моментів для початківців - є розуміння сутності структури алгоритмів та взаємна взаємодія компонентів програм, чим зумовлена потреба у нових, більш досконалих підходах щодо подання навчальних матеріалів.

Метою даної роботи є розробка системи транслювання коду між текстовим представленням та аналогічною графічною схемою, яка може бути застосована під час навчання на початкових етапах програмування для чіткого розуміння та наочного бачення алгоритму роботи коду.

Створення даної системи дозволяє знизити поріг входження у галузь, через можливість набуття певних навичок на більш ранніх етапах навчання, тобто це має прискорити та спростити процес навчання для програмістів-початківців.

Отже, метою роботи є створення допоміжного продукту, який може бути застосовано для освітніх цілей на початкових кроках навчання програмуванню та сприятиме формуванню чіткого бачення і розуміння роботи алгоритмів.

РОЗДІЛ 1

АНАЛІЗ ІСНУЮЧИХ ГРАФІЧНИХ РЕДАКТОРІВ

1. Загальні відомості

В даній роботі присутня специфічна термінологія та використано різноманітні технології. Для отримання повного розуміння щодо процесу розробки та функціонування клієнт-серверної системи трансляції програмного коду між графічним та текстовим представленням, у наступних пунктах описано технології, що є базисом для виконання роботи, а також більш детально розкрито поняття, що потребують тлумачення.

1.1. Алгоритм

Алгоритмом називається кінечна сукупність точно заданих інструкцій для вирішення певної задачі. Алгоритми наділено низкою доволі важливих властивостей, сутність яких більш детально розкрито у наступних пунктах :

- Скінченність. Завершення алгоритму є обов’язковим та має відбуватись після виконання скінченної кількості кроків. Метод обчислень - це процедура, яка має характеристики алгоритмів, але не має скінченності
- Дискретність. Визначений алгоритмом процес можна відокремити елементарними етапами, кожен з яких є кроком алгоритмічного процесу чи алгоритму [7].
- Визначеність. Усі кроки алгоритму мають бути чітко визначеними. Необхідні для здійснення дії кроки мають точно та однозначно визначатися для будь-якого можливого випадку.
- Вхідні дані. Для алгоритма наявна певна кількість, яка також може бути нульовою, вхідних даних. Вхідними даними називаються величини, що задано до початку роботи алгоритму або їх значення мають бути сформовані під час здійснення алгоритму.
- Вихідні дані. Величини з визначеним зв’язком із вхідними даними

називаються вихідними даними. Кількість їх значень варьюється від одного до декількох, якщо відсутність даних також вважати результатом.

- Ефективність. Ефективність алгоритму визначається достатньою простотою операторів для того, щоб їх чітке виконання могло відбутись у скінченний проміжок часу, за наявності лише олівця та аркушу паперу, якщо моделювати алгоритм, наприклад, у письмовому вигляді.
- Масовість. Алгоритмом має забезпечуватись можливість розв’язання кожної однотипної задачі за умови будь-яких вхідних даних з області, в якій алгоритм застосовується [8].

1.2. Веб-сторінка

Оскільки розробка програмного продукту відбуватиметься на базі веб-сторінки, є вкрай важливим повне розуміння того, що являє собою сутність веб-сторінок та як має здійснюватися робота з ними.

Веб-сторінкою називається інформаційний ресурс, доступ до якого відбувається через взаємодію з мережею World Wide Web за наявності певної адреси ресурсу. Вихідний код веб-сторінки зазвичай записується в форматі HTML, або XHTML. В процесі створенні веб-сторінки використовують, мову розмітки HTML, таблиці каскадних стилів CSS та мову програмування JavaScript, яка додає сторінці інтерактивність.

1.3. Протоколи та методи запитів

Доступ до веб-сторінки відбувається у Всесвітній мережі Інтернет. Для отримання інформації сторінок, є необхідним використання протоколу HTTP.

Протокол передачі гіпертексту (HTTP) - це протокол рівня додатків для розподілених, спільних, інформаційних систем гіпермедіа [9].

Протокол HTTP передбачає використання клієнт-серверної структури передачі даних. Клієнтською програмою формується запит, який потім

відправляється на сервер, після чого серверне програмне забезпечення обробляє цей запит і формує відповідь, яка у результаті передається клієнтові.

Завдання, яке традиційно вирішується за допомогою протоколу HTTP - обмін даними між призначеним для користувача додатком, що здійснює доступ до веб-ресурсів і веб-сервером. На даний момент саме завдяки протоколу HTTP забезпечується робота Всесвітньої павутини.

Існують наступні методи, за якими здійснюється робота протоколу HTTP:

- GET - означає отримання будь-якої інформації, що ідентифікується URI-запитом та не впливає на її зміст;
- POST - означає отримання будь-якої інформації, ідентифікованої Request-URI та може впливати на зміст ресурсу;
- OPTIONS - являє собою запит на інформацію про варіанти зв'язку, які є доступними в ланцюзі запитів або відповідей, ідентифікованих URI-запитом. Зазвичай описують комунікаційні параметри клієнта та сервера;
- HEAD - ідентичний GET, за винятком того, що сервер НЕ ПОВИНЕН повертати тіло повідомлення у відповідь;
- PUT - вимагає, щоб вкладений об'єкт зберігався під наданим URI-запитом;
- DELETE - вимагає, щоб сервер-джерело видалив ресурс, ідентифікований URI-запитом;
- TRACE - використовується для виклику віддаленої зворотної петлі на рівні запиту повідомлення;
- CONNECT - для використання з проксі-сервером, який може динамічно переходити в тунель;

Найбільш поширеними методами є GET та POST. Різниця між ними полягає в тому, що методом GET передаються параметри запиту разом з URL, а POST робить це в тілі запиту. Це означає, що фактично для метода GET є обмеження на довжину параметрів, зумовлене обмеженням на довжину URL за яким передається запит. Ця довжина варіюється в залежності від браузера, який використовується. Існує максимальне обмеження. Наприклад, для веб-браузеру від компанії Microsoft - Internet Explorer воно складає 2048 символів. Для інших браузерів зазвичай дозволяється більша кількість символів [10].

2. Аналіз існуючих рішень

2.1. code2flow

2.1.1. Огляд функціоналу

Розглянемо перший аналогічний інтернет-ресурс code2flow.com, що дозволяє миттєво транслювати вихідний текст та створювати його відображення у вигляді блок-схем.

За використання Code2flow, надається доступ до багатофункціональної платформи, спосіб дії якої полягає у розробці макетів блок-схем за витратою мінімального часу, а також з подальшою можливістю публікації проекту на інших ресурсах.

Вищезазначений додаток дозволяє створювати схеми для приватного використання або подальшого розповсюдження, тобто містить інструменти експорту. До того ж, має простий синтаксис. У code2flow присутня низка можливостей для інтегрування даної системи з іншими.

Зображення (Рис. 1.1) демонструє інтерфейс code2flow, а також дозволяє дійти висновку, що цей додаток не є повною мірою пристосованим саме для роботи з кодом. Тобто сервіс code2flow використовує лише базові фігури для демонстрації алгоритму.

Система налаштувань додатку представлена додатковими опціями, що дозволяють також редагувати зовнішні характеристики діаграми. Серед них є наступні:

- Налаштування тексту умов для оператора розгалуження;
- Зміна теми додатку;
- Зміна масштабу зображення;
- Зміна напрямку графу блок-схеми;
- Розмір шрифту.

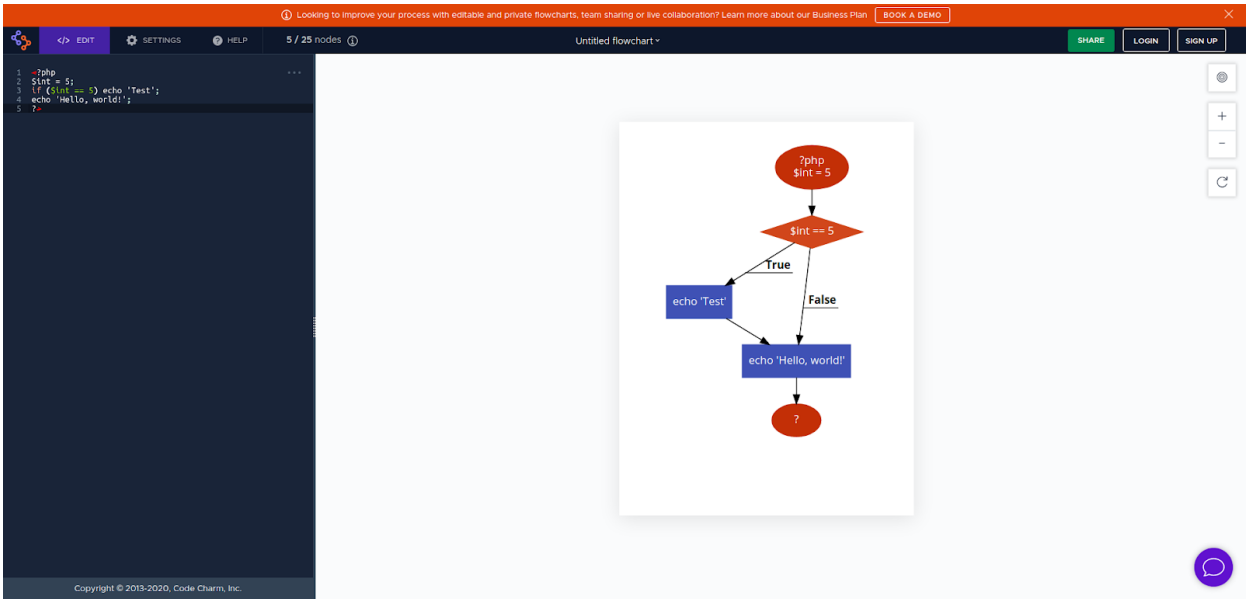


Рис. 1.1 Інтерфейс редактора коду “code2flow”

2.1.2. Переваги та недоліки

Переваги:

- Зручний та зрозумілий для користувача інтерфейс;
- Висока швидкість, оскільки для відображення алгоритму не має відбуватися перезавантаження сторінки;
- Відсутність потреби у додаткових діях для перетворення тексту в блок-схему;
- Часткове ознайомлення з функціоналом сайту, за умови використання безкоштовної версії;

- Даний сервіс представлений у якості веб-ресурсу, отже його використання можливе у більшості ОС;
- Збереження коду при закритті сторінки сайту;

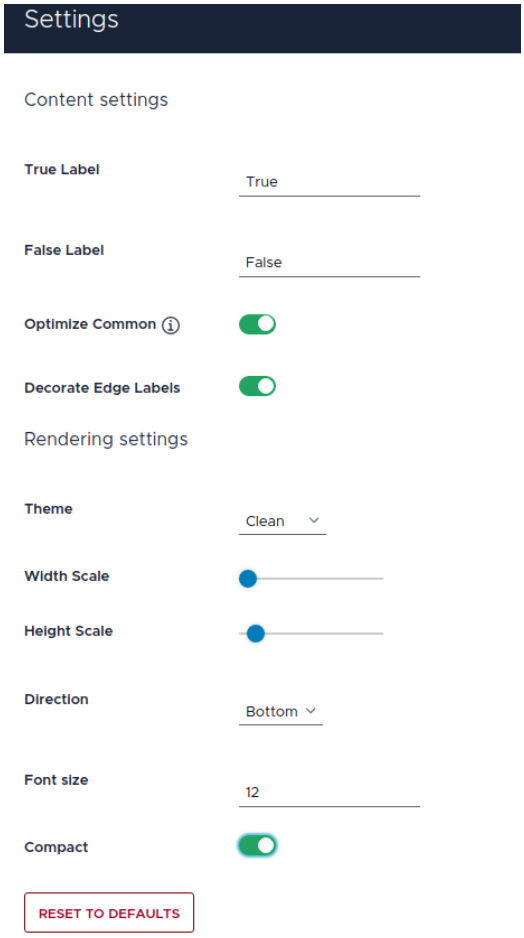


Рис. 1.2 Панель налаштувань

Недоліки:

- Безкоштовне використання сайту можливе за умови обмеженого функціоналу, оскільки повний перелік функцій передбачає придбання платної підписки, ціна якої починається від $10 на місяць;
- Генерация блок-схем відбувається не завдяки трансформації коду, а з формування псевдокоду. Тобто даний сервіс не пристосовано для певної мови програмування або декількох мов. Отже, у окремих випадках використання сервісу code2flow здатне сприяти отриманню помилкового алгоритму. Наприклад, за умови використання функцій

виводу інформації;

- Також слід зазначити, що відбувається трансформування коду в єдиному порядку, тобто відсутня можливість зворотної генерації коду з блок-схеми;
- Відсутня можливість пересування блоків у схемі за використання методу Drag-and-drop.

2.2. Aivosto Visustin

2.2.1. Огляд функціоналу

Visustin - це автоматизована програма, орієнтована на створення блок-схем для використання розробниками програм і авторами документів.

Visustin виконує зворотний інжиніринг наданого вихідного коду, з метою його перетворення у відокремлені блоки та подальшого формування блок-схеми або діаграми діяльності UML (Activity Diagram).

Visustin здійснює вищезазначені операції за допомогою зчитування операторів if та else, операторів циклу та операторів переходу, і - далі цілковито у автоматичному режимі - створення блок-схеми.

Додаток Visustin користується доволі великою популярністю. На офіційній сторінці наявний розділ, що містить відгуки. Користувачі ресурсу вважають, що дана програма допомагає заощадити час та дозволяє уникнути рутинну роботу з відлагодження програм.

Також на офіційній сторінці продукту представлена інформація щодо клієнтів, які використовують можливості Visustin. Серед них є такі відомі компанії, як: BMW, Cisco, Deloitte, Google, IBM, NASA, NVIDIA, та інші.

2.2.2. Переваги та недоліки

Переваги:

- Додаток може обробляти близько 51 мови, та деякі підвиди мов;
- Є можливість експорту блок-схеми в Microsoft Visio та Microsoft Word;

- Розробниками було додано приклади використання для більшості мов, що підтримуються додатком. Це дозволяє початківцям ознайомитись з синтаксисом мов, та побачити структуру коду через візуалізацію;
- Є доволі гнучка система налаштувань, за допомогою якої можна налаштувати наступні елементи інтерфейсу: робочий простір, коментарі, вигляд блок-схем;

Недоліки:

- Дана програма існує виключно у платному варіанті, ціна якого починається від $149;

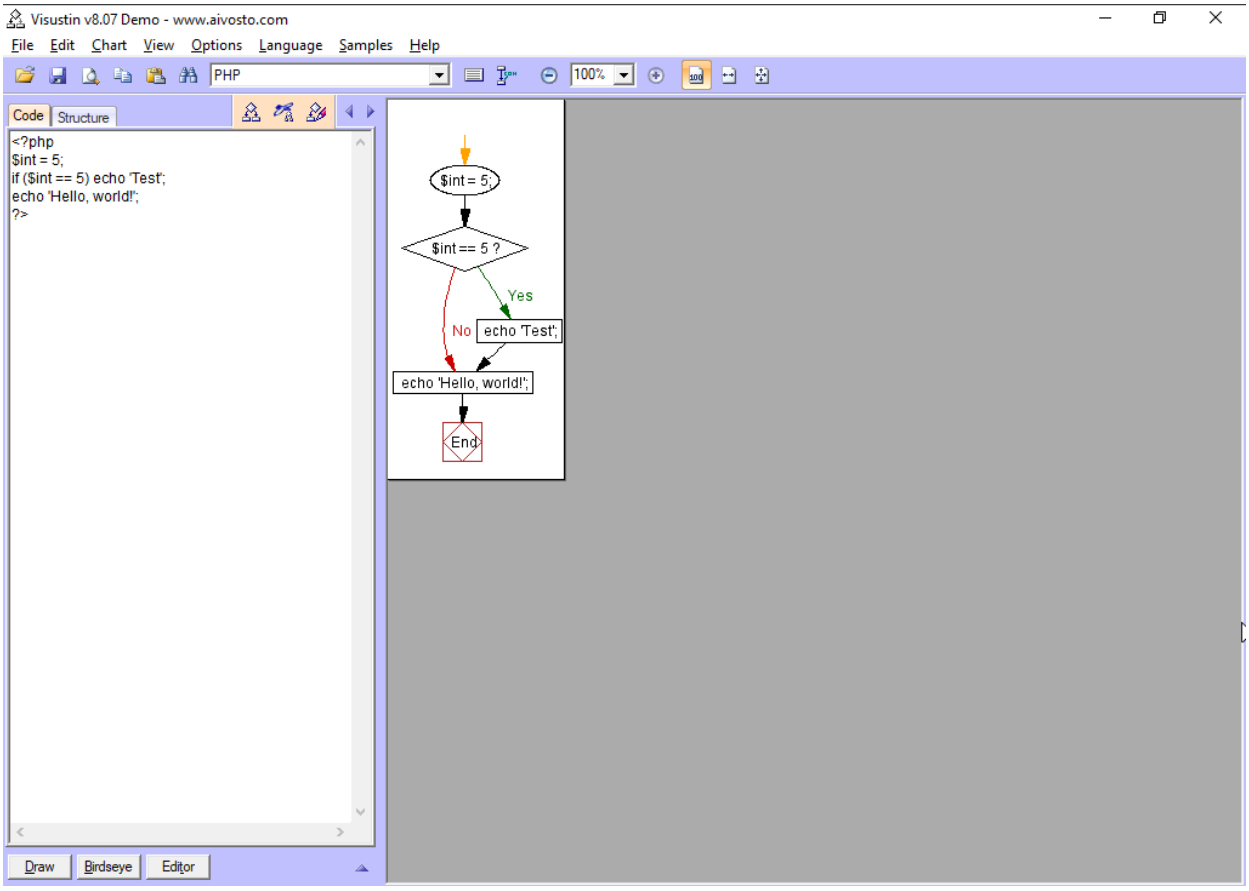


Рис. 1.3 Головний екран додатку “Aivosto Visustin”

- Інтерфейс програми є занадто примітивним та не є сучасним, що робить її менш зручною;
- Даний додаток підтримується лише на ОС Windows, або за допомогою емуляторів чи віртуальних машин що підтримуюсь ОС Windows;

- Можливість рухати блоки у цьому додатку присутня, але лише в спеціальному режимі та без повного функціоналу у тестовій версії програми;
- Не відбувається диференціації відображення блоків з функцією виводу інформації від звичайних операційних блоків;
- Відсутня підтримка конструкції циклів. В даному додатку дана можливість реалізована за допомогою використання звичайної конструкції умови, але у вигляді післяумови;

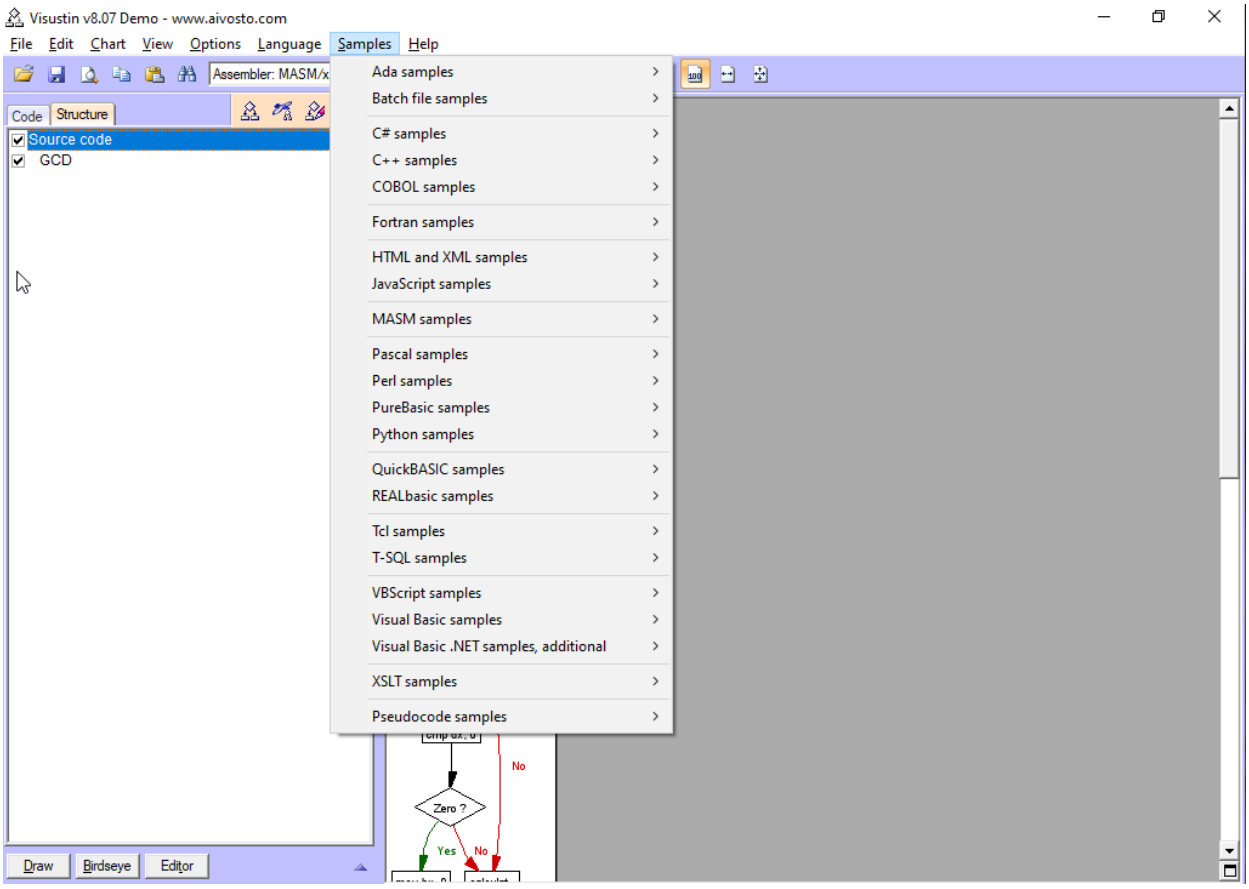


Рис. 1.4 Вибір прикладів з кодом

3. flowgorithm

3.1. Огляд функціоналу

Flowgorithm - це продукт, що має низку графічних інструментів для виконання користувачами програм за допомогою використання блок-схем.

Особливістю Flowgorithm є підхід, що орієнтовано саме на підкреслення

алгоритму, а не акцентування на синтаксисі конкретної мови програмування [2].

3.1.1. Переваги та недоліки

Переваги:

- Додаток має зручний і зрозумілий інтерфейс, крім безпосередньо побудови блок-схем;
- Є можливість зміни мови додатку;
- Додаток є безкоштовним;
- Діаграма може бути перетворена на декількох мовах програмування;

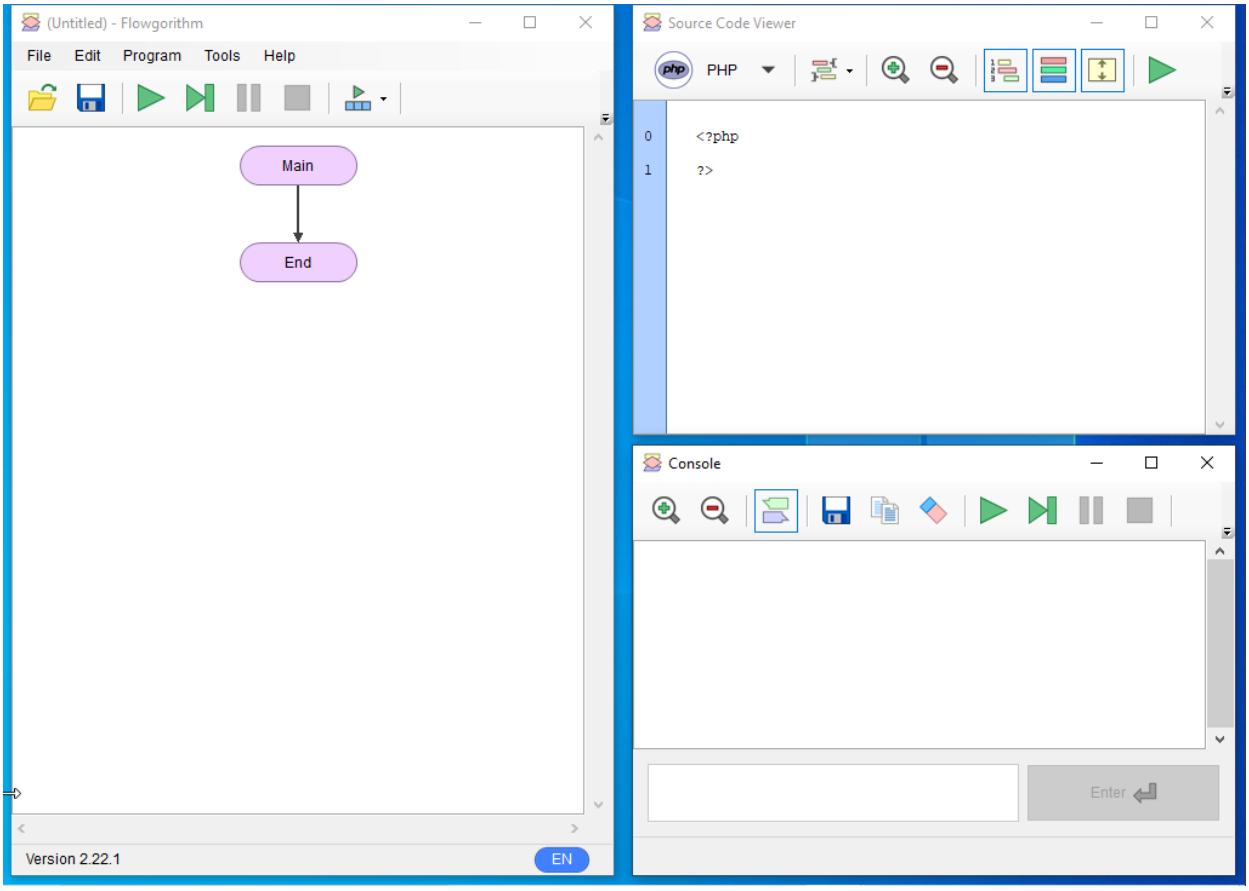


Рис. 1.5 Головне вікно програми “Flowgorithm”

- Є функція зміни теми, а саме зміна теми фону, та блоків. Окрім вибору зі збудованих тем, існує можливість меню створення власної теми та оформлення блоків;
- Програма є зрозумілою та простою у роботі;

Недоліки:

- На освоєння функціоналу програми потребується багато часу та детальне вивчення документації;
- Відсутня можливість редагувати безпосередньо код;
- Використання додатку можливо лише на ОС сімейства Windows;

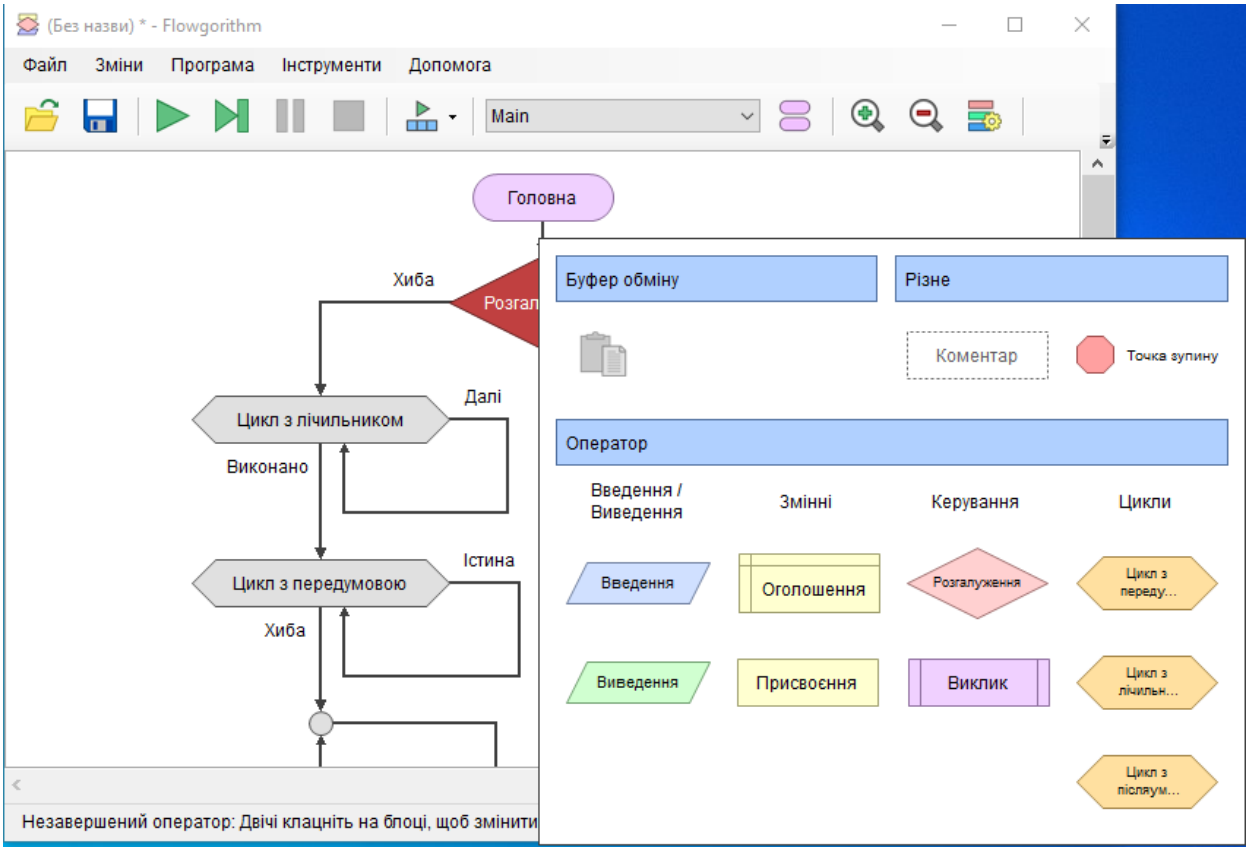


Рис. 1.6 Вікно вибору оператора

ВИСНОВОК РОЗДІЛУ 1

В даному розділі було розглянуто програми, мета функціонування яких полягає у взаємодії роботи коду та блок-схем. Основними перевагами цих програм є: висока швидкість; кроссплатформенність, за рахунок використання веб-технологій; збереження коду при закритті сторінки сайту; підтримка багатьох мов програмування; можливість експорту блок-схеми в Microsoft Visio та Microsoft Word; гнучка система налаштувань інтерфейсу та відображення блок-схем; можливість зміни мови додатку.

Також було визначено певні недоліки, що не дозволяють назвати будь-яку з вищезазначених програм універсальною, а саме: ціна продукту; відсутня можливість зворотної генерації коду з блок-схеми; відсутня можливість пересування блоків у схемі та використання методу Drag-and-drop; Підтримка лише ОС Windows; не відбувається диференціації відображення блоків; відсутня підтримка деяких конструкції, як, наприклад, цикли; на освоєння функціоналу програми потребується багато часу та детальне вивчення документації;

Отже, висновком даного аналізу можно вважати доцільність створення програмного продукту, у якому мають бути враховані нюанси, що дозволять удосконалити переваги та мінімізувати недоліки аналогічних продуктів.

Основними цілями при розробці клієнт-серверної системи трансляції програмного коду між графічним та текстовим представленням, є створення зручного функціоналу, за дотримання простоти та доступності, що робитиме програму придатною для якнайбільшого числа користувачів.

РОЗДІЛ 2

СПОСОБИ ОПИСУ ПРОГРАМНИХ АЛГОРИТМІВ

1. Блок-схема програми

Блок-схемою називається поширений тип схем, а саме графічних моделей, що описують алгоритми або процеси. Зазвичай, блок-схеми зображуються у вигляді орієнтованого графу. Опис окремих кроків алгоритму відбувається у вигляді вузла графа різної форми. Ці кроки об’єднуються між собою лініями, що представляють собою ребра графу, а також вказують напрямок послідовності.

2. Базові структури блок-схем

Відповідно до положень теореми Бьома-Якопіні, будь-який виконуваний алгоритм може бути реалізовано за умови використання трьох конкретних керуючих структур. Цими структурами є послідовне виконання, розгалуження, повтори (цикли).

Будь-який алгоритм може бути описано за допомогою використання лише трьох даних структур. На даний момент для спрощення процесу роботи з блок-схемами та адаптації теореми для застосування на практиці, вищезазначені керуючі структури було поділено на більш дрібні структури.

Далі будуть продемонстровані базові структурні схеми, за допомогою яких можна описати будь-який алгоритм. На практиці не використовують саме цей елементний базис, оскільки це лише теоретичне частина алгоритмів.

2.1. Базова структура “Слідування”

Структура “Слідування” (Рис 2.1) є алгоритмічною структурою, що забезпечує отримання результату шляхом виконання одноразових послідовних дій незалежно від даних та результату. В подібних структурах дії виконуються послідовно, тобто лінійно.

2.2. Базова структура “Розгалуження”

Структура “Розгалуження” являє собою вибір дій, що відбувається у разі чіткого виконання або невиконання заданої умови.

Однією з умов використання розгалуження є те, що незалежно від виконання умови та обраного шляху, алгоритм веде до загального виходу, отже робота алгоритму триватиме далі.

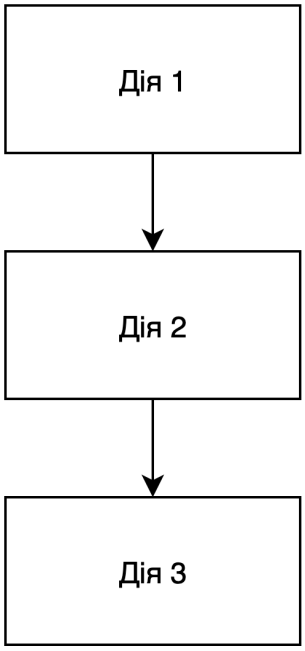


Рис. 2.1

Розгалуження бувають наступних типів:

- Повні розгалуження - це розгалуження, в яких певні дії є визначеними, як у разі виконання, так і за невиконання умови;
- Неповні розгалуження - це розгалуження, в яких дії визначено у разі виконання або невиконання умови. Тобто лише за впровадження одного з варіантів.

Структура розгалуження існує в чотирьох основних варіаціях:

- якщо то;
- якщо то-інакше;
- вибір;

- вибір-інакше.

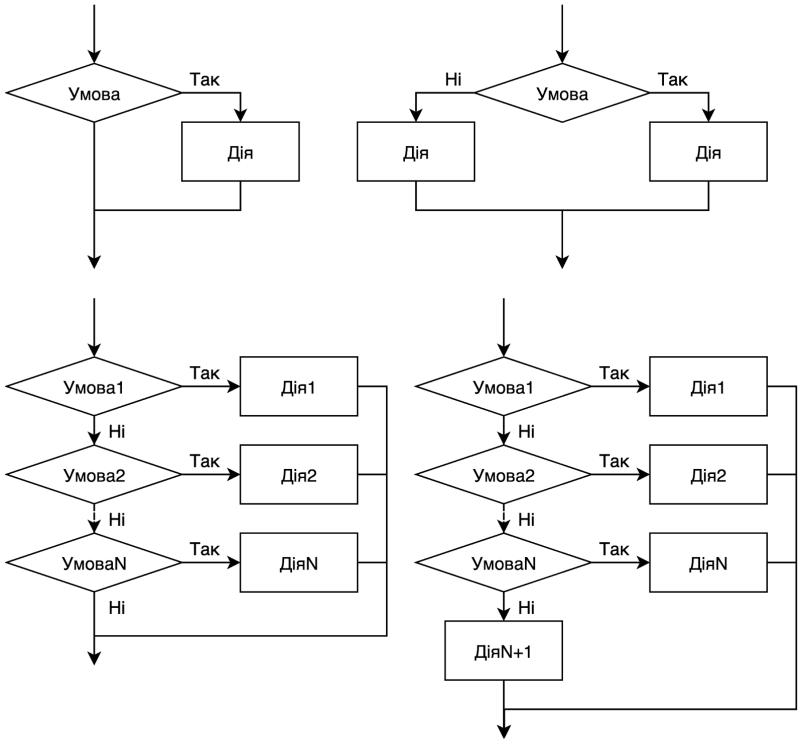


Рис. 2.2

2.3. Базова структура “Повторення”

Структура “Повторення” - це різновид базової структури, в якій передбачається виконання певної операції, або серії операцій задану кількість разів.

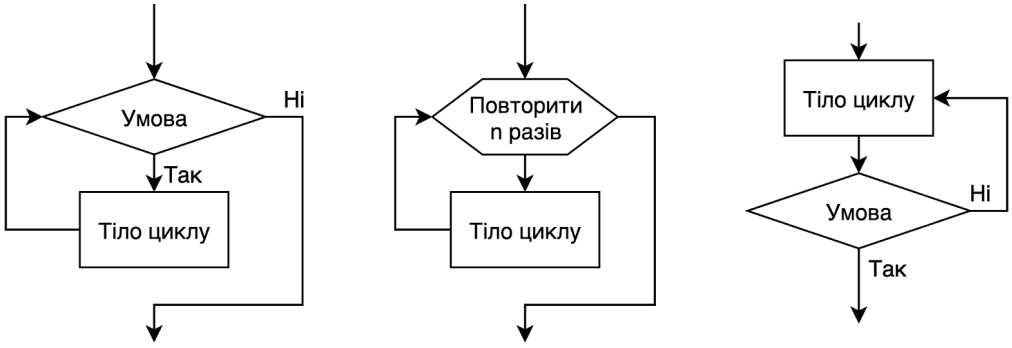


Рис. 2.3

Використання повторення дозволяє в короткий спосіб описати виконання великої кількості операції. Певна сукупність дій, що виконуються,

називаються тілом циклу. На зображенні (рис. 2.3) наведено основні різновиди циклів.

3. Елементи блок-схем

За державним стандартом ГОСТ 19.701-90 (ISO 5807-85) регламентуються нормативи щодо того, яким має бути вигляд блок-схем, а також нормативи щодо порядку їх застосування [3].

Найбільш поширені варіанти нормативно регламентованих елементів блок-схем, що застосовуються в даному програмному продукті, продемонстровано на зображеннях (Рис. 2.4-11), які наведено у нижчезазначених пунктах розділу.

3.1. Процес

Символ відображає функцію обробки даних будь-якого виду (виконання певної операції або групи операцій, що призводить до зміни значення, форми або розміщення інформації або до визначення, за яким з декількох напрямків потоку слід рухатися).

Рис. 2.4

3.2. Термінатор

Символ відображає вихід в зовнішнє середовище і вхід із зовнішнього середовища (початок або кінець схеми програми, зовнішнє використання і джерело або пункт призначення даних).

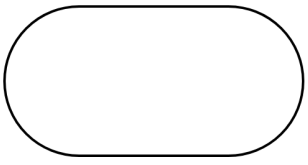


Рис. 2.5

3.3. Дані

Символ відображає введення в програму, або виведення з неї даних. Носій, джерело, або отримувач даних не є визначеними. У програмах, що працюють у консолі найчастіше мається на увазі введення користувача, або виведення інформації у консоль.

Рис. 2.6

3.4. Функції

Символ відображає завчасно визначену функцію, яка складається з однієї або декількох операцій, які визначено в іншому місці.

Рис. 2.7

3.5. Підготовка

Символ відображає підготовку з метою впливу на наступну функцію. Такий символ використовують при ініціалізації найпростішого циклу. Також це може бути символом зміни перемикача або модифікації індексного регістру.

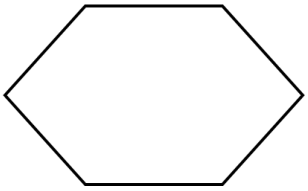


Рис. 2.8

3.6. Умова

Символ відображає рішення або функцію перемикача типу, що має один

вхід і ряд альтернативних виходів, лише один з яких може бути активізований після обчислення умов, визначених всередині цього символу.

Відповідні результати обчислення можуть бути записані поряд з лініями, що відображають ці шляхи.

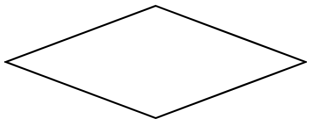


Рис. 2.9

3.7. Лінія

Основний символ з’єднання елементів в блок-схемах - лінії.

Лінія являє собою представлення переходу до наступного блоку, а також потік даних. В коді ліній ніяк не побачити, адже символ між рядками є невидимим.

Рис. 2.10

3.8. Коментарі

Не менш важливим елементом кода є коментарі до нього. Вони повинні обов’язково бути присутніми при трансляції.

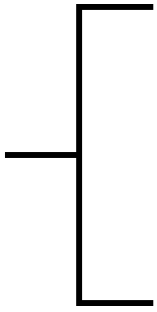


Рис. 2.11

4. Приклади трансляції коду в блок-схеми

У попередньому розділі було продемонстровано складові блок-схем та розкрито їх сутність, отже на зображеннях (Рис. 2.12-16), що наведено у

даному пункті, наочно показано яким чином відбувається транслювання коду у графічний вигляд, тобто його трансформація у візуальне представлення - блок-схеми.

4.1. Покроковий алгоритм

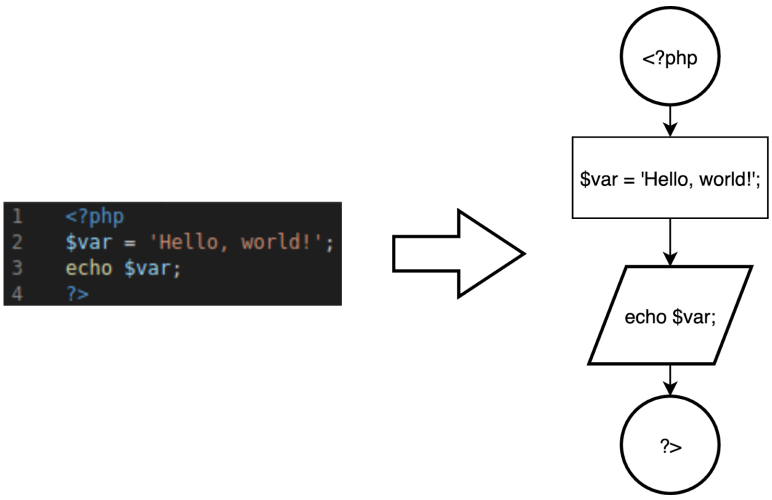


Рис. 2.12

4.2. Алгоритм з умовою

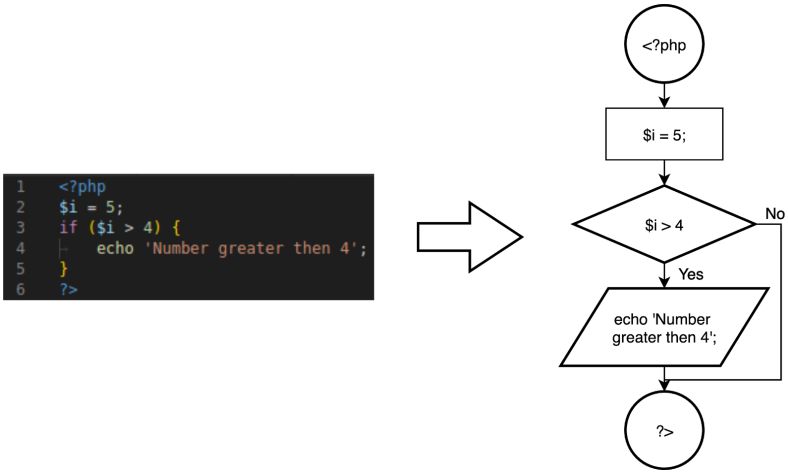


Рис. 2.13

4.3. Алгоритм з циклом

```php
<?php
for ($i = 0; $i < 10; $i++) {
    echo 'Lorem ipsum';
}
?>
```

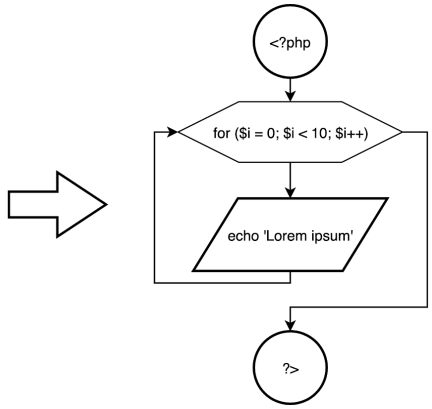


Рис. 2.14

4.4. Алгоритм з вибором

```php
<?php
$age = 18;
switch ($age) {
    case 17:
        echo 'Born in 2003';
        break;
    case 18:
        echo 'Born in 2002';
        break;
    case 19:
        echo 'Born in 2001';
        break;
    default:
        echo 'Have no idea when you where born';
        break;
}
?>
```

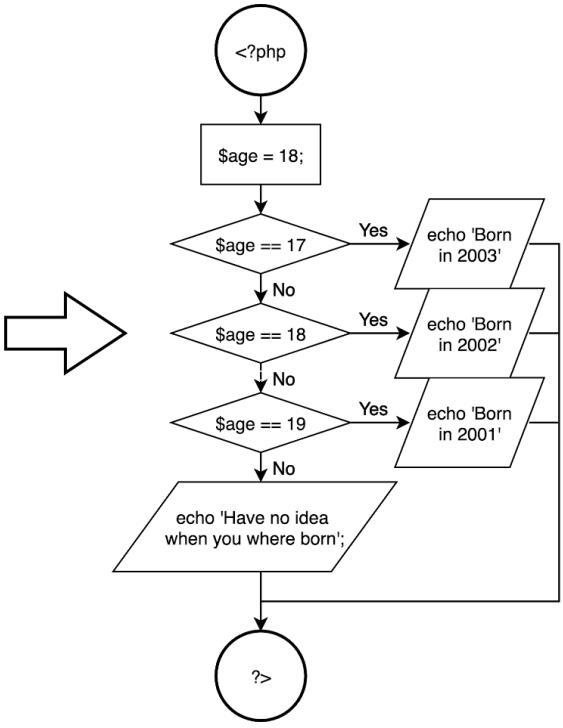
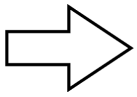


Рис. 2.15

4.5. Алгоритм з післяумовою

```php
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```



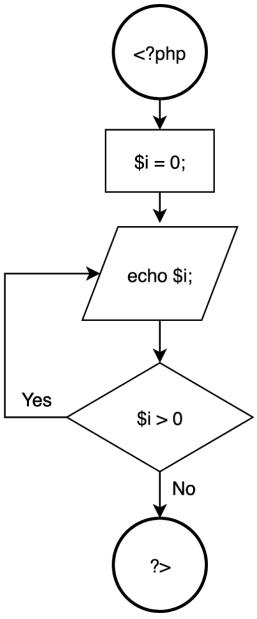


Рис. 2.16

ВИСНОВОК РОЗДІЛУ 2

В даному розділі було розглянуто види блок-схем та їх складові компоненти, а також більш детально розкрито сутність та описано призначення кожного елементу.

Згідно з положеннями теореми Бьома-Якопіні для побудови алгоритму вважається достатньою наявність трьох фігур.

Оскільки практичне використання не обмежуються даними фігурами та включає до себе елементний базис, подібна концепція не дозволяє повною мірою відобразити алгоритми у додатку, створення якого є метою даної роботи. Для детальнішого відображення алгоритмів, введена більша кількість та різноманітність фігур.

Зазвичай, види наведених у додатку елементів є достатніми для побудови більшості алгоритмів. Але через існування специфічних фігур, використання яких є притаманним для дуже вузьких спеціалізацій, у роботі також було наведено приклади коду та відповідні до кода блок-схеми.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

1. Постановка задачі

Завданням даної роботи є створення програмного середовища з графічним інтерфейсом, який дозволяє користувачеві транслювати код між графічним та текстовим представленням, зберігати та завантажувати створені алгоритми. Перевірка синтаксису введеного користувачем коду не відбувається.

Клієнтська частина системи надає користувачеві можливість вводу коду, редагування блок-схем, та надсилання коду на сервер. Також на стороні користувача відбувається генерація та відображення блок-схеми з масиву даних.

Масив даних система має отримувати за рахунок того, що відбувається запит зі сторони користувача, в результаті якого код повертається сервером. Після створення користувачем запиту, серверна частина системи має отримати код, що надсилається, а далі відбувається розробка програмного коду на токени та генерування масиву даних, який, в якості відповіді на запит, надсилається користувачеві.

2. Вибір платформи

Першим кроком у розробці клієнт-серверної системи трансляції програмного коду між графічним та текстовим представлення має вибір платформи, що дозволяє продемонструвати якомога більше переваг додатку, а також має бути найбільш зручною для досягнення мети у найшвидший і найкращий спосіб. На даний момент, спостерігається доволі велика різноманітність ОС для ПК. Найбільш поширенішими, а отже платформами з вищим рівнем адаптації та більшим переліком можливостей є наступні: Microsoft Windows, OS X, Linux.

На зображенні (Рис. 3.1) продемонстровано статистичну вибірку, яка містить відсоткове співвідношення щодо вибору, тобто популярності використання, операційної системи користувачами. Виходячи з отриманої інформації, можна зробити висновок, що найбільш популярною з них є ОС Microsoft Windows.

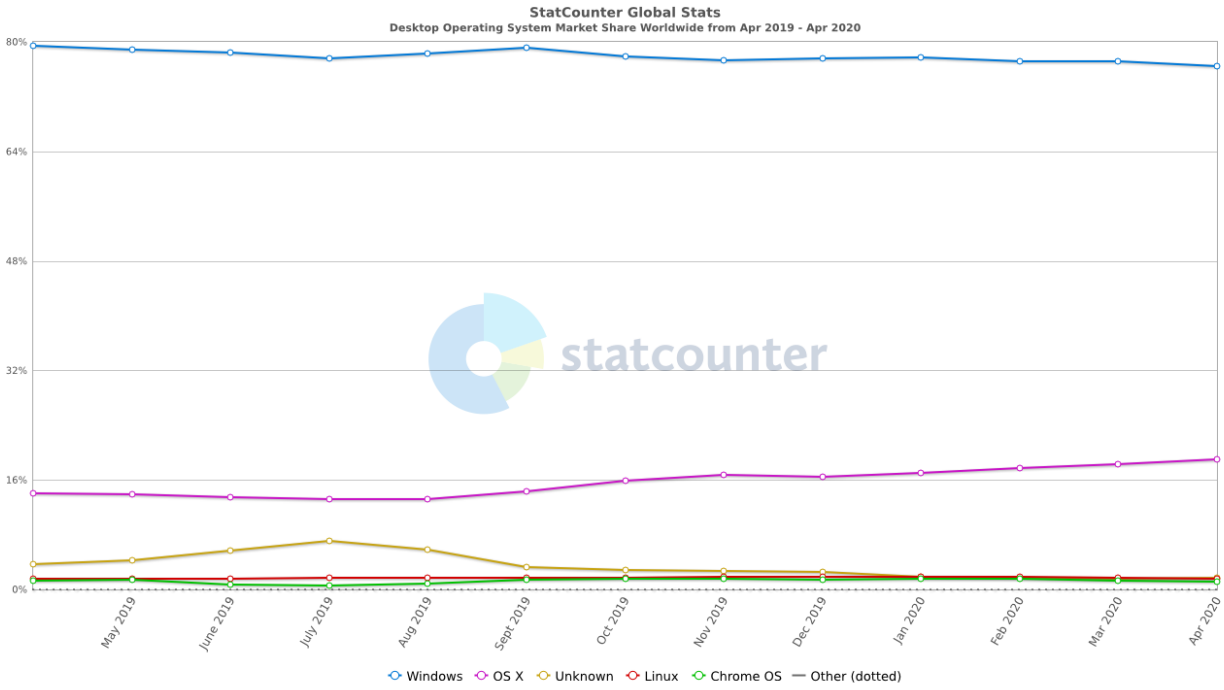


Рис. 3.1

Для кожної ОС наявні певні переваги та недоліки. Для того, щоб визначити найбільш доречну платформу для розробки програмного продукту, далі пропонується розглянути особливості найбільш поширених ОС:

Microsoft Windows:

- Вибаглива до апаратної частини;
- Найбільш вразлива до вірусів та інших атак;
- Проста для встановлення для пересічного користувача;
- Оскільки Microsoft Windows є найбільш поширеною, кількість різноманітного програмного забезпечення є найбільшою;
- Простий та зрозумілий інтерфейс;

OS X:

- Найчастіше розробку програмного забезпечення для цієї ОС, необхідно робити використовуючи OS X;
- Найбезпечніша операційна система. OS X було переписано з нуля, отже пошук та виявлення вразливостей для системи ускладнилося;
- Достатня кількість програмного забезпечення, встановлення якого можливе лише з офіційного магазину AppStore;

Linux:

- Найменш вибаглива до апаратної частини;
- Оскільки ОС підтримується в тому числі користувачами, то нові вразливості та помилки виправляються доволі швидко;
- Встановлення певних версій цієї ОС, може бути складним навіть для досвідчених користувачів;
- Наявною є доволі велика кількість доступного програмного забезпечення. Крім того, також існують емулятори, які дозволяють використовувати програмне забезпечення для ОС Windows;

Отже, можна зробити висновок, що кожна ОС в чомусь є кращою за інші та водночас має низку недоліків, що не дозволяють виділити лише одну як найкращу. Створення програмного продукту лише для певної ОС, обмежує його доступність, що протирічить базовим засадам ідеї розробки. Додаток має допомагати спростити процес навчання та покращити розуміння алгоритмів для якомога більшого числа користувачів. Враховуючи той факт що згідно зі статистикою доволі популярного серед IT-суспільства интернет-ресурсу “Stack Overflow” за 2019 рік [4] більшість користувачів роблять вибір на користь ОС Linux. Отже питання з визначення операційної системи з найбільшою кількістю користувачів, в яких є потреба у клієнт-серверній системі трансляції програмного коду між графічним та

текстовим представленням є доволі неоднозначним.

До того ж, написання додатку водночас для декількох ОС має займати майже вдвічі більше часу в порівнянні з розробкою лише для однієї платформи. Відповідно до цього було вирішено знайдено компроміс у вигляді ідеї багатоплатформної розробки.

Це рішення вплинуло на кількість варіантів, яка суттєво зменшилася аби задовольнити потребу у адаптивності, поширеності та доступності клієнт-серверної системи трансляції програмного коду. Ліпшим з варіантів, що залишилися є розробка на мовах, що інтерпретуються незалежно від ОС або веб-технології. Враховуючи вищезазначене вибір платформи зупинився на Web-технологіях.

3. Вибір мов програмування

Для створення програмного продукту було обрано веб-середовище, отже вибір мов відбувся з найбільш поширених та популярних мов у веб-просторі. Оскільки використання “чистих” мов не є обов'язковим, огляд буде розповсюджуватись окрім мов, також на інші популярні технології. Інтерес полягає в налаштуванні робочого середовища. Це має бути зручним та зрозумілим для користувача, тому використання додаткових бібліотек або сторонніх кодів вважається допустимим.

3.1. Front-end та back-end

Взагалі, терміни фронт-енд (англ. front-end) та бек-енд (англ. back-end) з’явилися внаслідок розвитку програмної інженерії для розподілу відповідальності між внутрішньою реалізацією та зовнішнім представленням.

Front-end являє собою клієнтську сторону програмного продукту у вигляді інтерфейсу користувача до програмно-апаратної частини сервісу.

Backend - це програмно-апаратна частина сервісу, яка для надання своєї функціональності реалізує API, що використовує front-end. Таким чином front-end розробнику не потрібно знати особливостей реалізації сервера, а back-end розробнику - реалізацію front-end [5].

3.2. Огляд front-end технологій

Оскільки обраною платформою стало веб-середовище, то відображення інтерфейсу буде відбуватись у веб-браузері.

Браузер - це прикладне програмне забезпечення створене для перегляду сторінок, змісту веб-документів, комп'ютерних файлів і їх каталогів; управління веб-додатками; а також для вирішення інших завдань. У глобальній мережі браузери найчастіше використовують для запиту, обробки, маніпулювання і відображення змісту веб-сайтів.

Багато сучасних браузерів також можуть використовуватися для обміну файлами з серверами FTP, а також для безпосереднього перегляду змісту файлів багатьох графічних файлів.

3.2.1. HTML + CSS + JS

Офіційно перші документації з’явилися 1992. З того часу зв’язка HTML з CSS та JS є стандартом для веб-сторінок. Якщо подивитись на принцип роботи веб-сторінок, то можна побачити, що всі інші технології так чи інакше надсилаються чи зводяться саме до таких мов, оскільки це мови які за замовчуванням мають оброблятися всіма браузерами. А отже використання саме цього набору технологій дозволяє зменшити час на генерування та обробку коду. З іншого погляду використання різних для платформи технологій значно збільшує час на розробку продукту, оскільки виконується багато рутинної роботи на кшталт написання базової розмітки сайту, найпримітивніших таблиці стилів, підключення всіх скриптів, тощо.

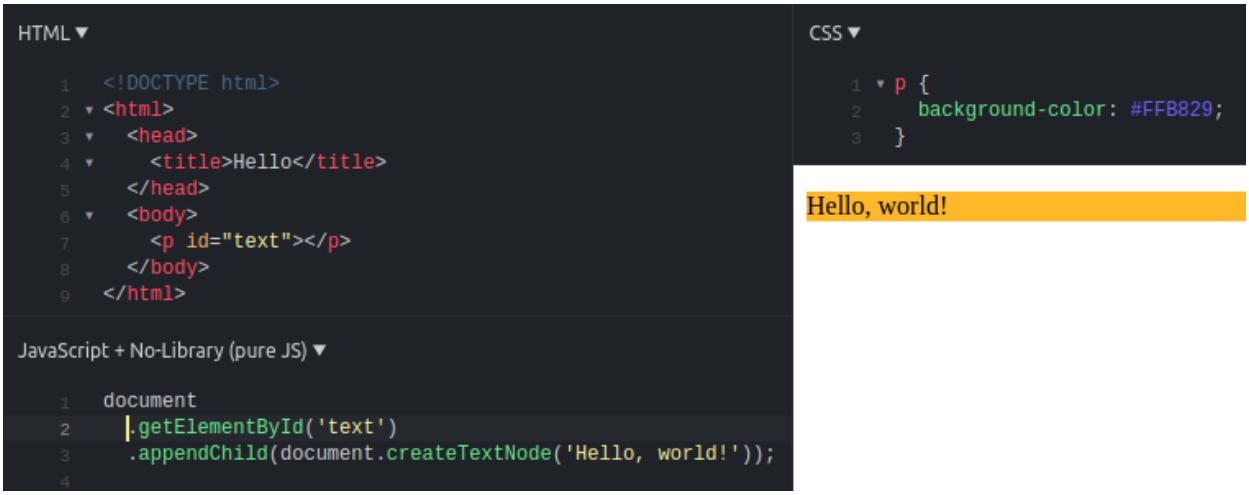


Рис. 3.2 Приклад HTML, CSS та JavaScript сторінки

3.2.2. jQuery

jQuery - це швидка та стисла бібліотека JavaScript, створена Джоном Ресігом у 2006 році з девізом: пишіть менше, робіть більше. jQuery спрощує обробку документів HTML, обробку подій, анімацію та взаємодію Ajax для швидкої веб-розробки. Тобто jQuery є інструментарієм JavaScript, призначеним для спрощення різних завдань, написання меншого обсягу коду.

Далі надається перелік найбільш важливих із основних функцій, які підтримує jQuery:

- Маніпуляція з DOM: jQuery спростив вибір елементів DOM, узгодження їх та зміну їх вмісту за допомогою перемикача з відкритим вихідним кодом браузера під назвою Sizzle;
- Обробка подій: jQuery пропонує зручний спосіб зйомки різних подій, таких як натискання на посилання, без необхідності збільшення HTML коду для обробки подій;
- Підтримка Ajax: jQuery допомагає розробити адаптивного та багатофункціонального сайту за допомогою технології Ajax;
- Анімації: jQuery поставляється з великою кількістю вбудованих анімаційних ефектів, які можна використовувати на веб-сайтах;

- Легка вага: jQuery - це легка бібліотека - розміром близько 19 КБ (мінімізована та gzipped);
- Підтримка крос-браузера: jQuery має підтримку крос-браузера і добре працює в IE 6.0+, FF 2.0+, Safari 3.0+, Chrome і Opera 9.0+ [6].

Деякі з цих переваг мають допомогти у розробці клієнт-серверної системи трансляції програмного коду між графічним та текстовим представленням, тому їх варто розглянути більш детально.

Ajax - це технологія асинхронного JavaScript та XML, що дозволяє виконувати обмін даних з веб-сервером “у фоновому режимі”. В результаті чого при оновленні даних, веб-сторінка не перезавантажується, що дає значний приріст у швидкості сайту та додає інтерактивності. На рисунку 3.3 зображений приклад сторінки з рисунку 3.2, але з використанням бібліотеки jQuery.

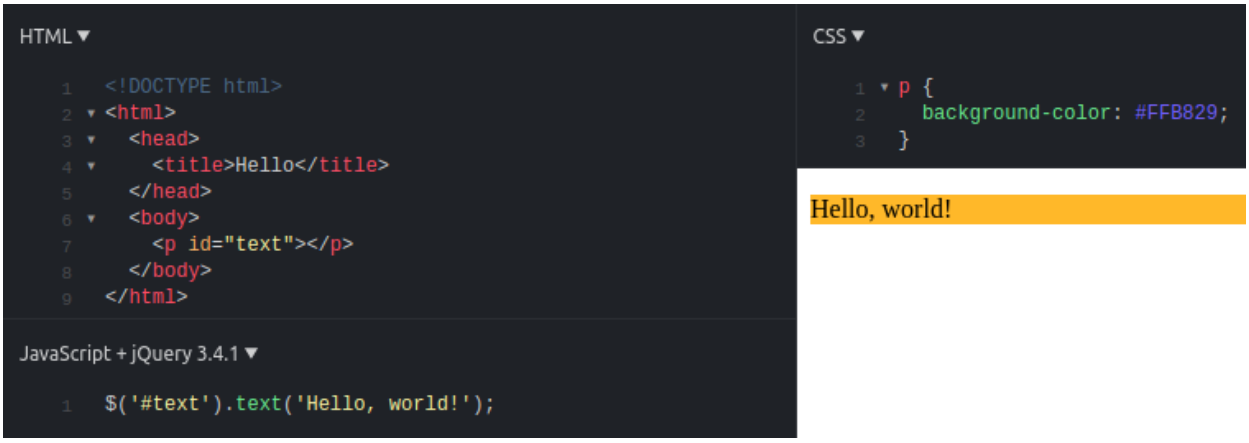


Рис. 3.3 Приклад сторінки з jQuery

В даному проекті jQuery є незамінним компонентом, оскільки для економії часу надсилання коду відбувається за допомогою технології Ajax, а отже без перезавантаження сторінки. Також jQuery дає можливість зручної роботи з елементами використовуючи звичні CSS-селектори.

3.2.3. Bootstrap

Однією з найпопулярніших UI бібліотек на сьогоднішній день є

Bootstrap. Розроблена компанією Twitter бібліотека, створена для спрощення процесу написання веб-сторінки. Компоненти Bootstrap складаються з готових HTML та CSS - шаблонів для оформлення типографії, форм, кнопок, та певного числа інших елементів сайту. Також важливою особливістю фреймворку є зручна система адаптивної верстки. Це означає, що у простий спосіб можна створити та налаштувати сайт для того, щоб він відображався коректно на всіх пристроях, незалежно від розмірів екрану.

В створенні клієнт-серверної системи трансляції програмного коду між графічним та текстовим представленням бібліотеку Twitter Bootstrap використано для оформлення зручного користувацького інтерфейсу.

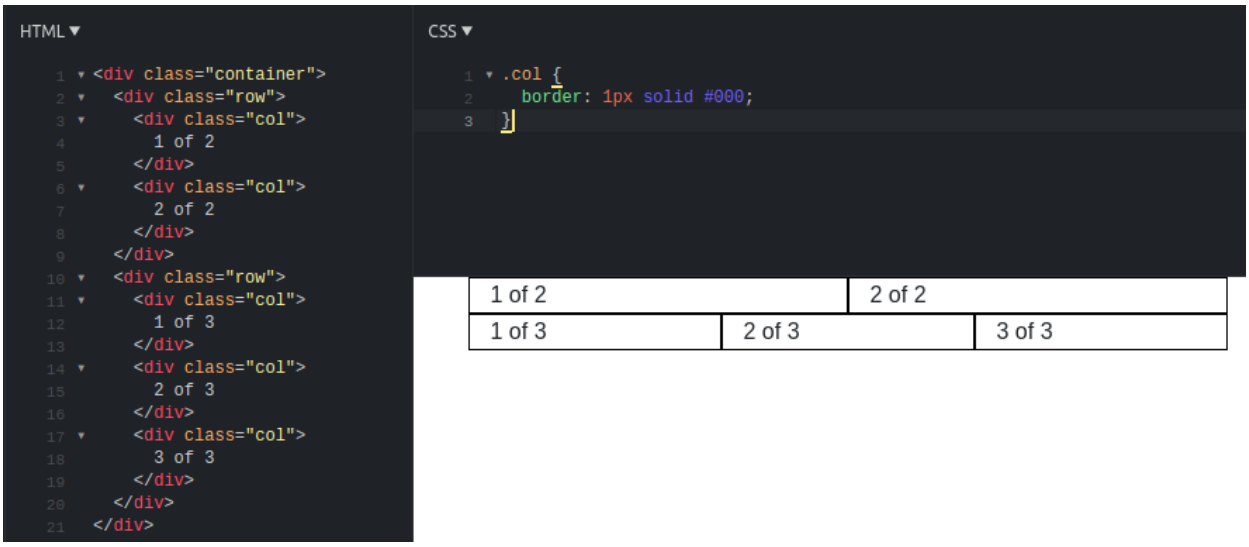


Рис. 3.4 Приклад сторінки з використанням Twitter Bootstrap

3.2.4. GoJS

GoJS - це бібліотека JavaScript і TypeScript для побудови інтерактивних діаграм і графіків. GoJS дозволяє будувати всілякі діаграми та графіки для своїх користувачів, від простих блок-схем та органічних діаграм до специфічних промислових діаграм, діаграм SCADA та BPMN, медичних діаграм, таких як генограми, діаграми моделювання спалахів тощо.

GoJS дозволяє легко створювати діаграми JavaScript із складних вузлів, посилань та груп за допомогою налаштованих шаблонів та макетів.

GoJS пропонує багато вдосконалених функцій для інтерактивності користувачів, таких як:

- Перетягуванн;
- Копіювання та вставка;
- Редагування тексту на місці;
- Підказки;
- Контекстні меню;
- Автоматичні макети;
- Шаблони;
- Прив'язка даних та моделі;
- Управління станом транзакцій та скасування;
- Палітри;
- Огляди;
- Обробники подій;
- Команди;
- Розширені інструменти для користувацьких операцій;
- Анімації з можливістю налаштування.

В даному проекті бібліотека GoJS використовується для відображення блок-схеми. Серед інших схожих фреймворків GoJS вирізняється наявністю можливості редагувати блок-схеми в дуже простий спосіб.

3.3. Огляд back-end технологій

Для створення серверної частини, у якій, власне, і відбуватиметься обробка коду і створення масиву даних. Використання бази даних на цьому кроці не є необхідним, але можливе використання для вводу системи користувачів, збереження коду для подальшої передачі посилання на цей код, тощо.

Оскільки все що необхідно на даному етапі - це парсинг коду з певної

мови у масив даних, то вибір мови був вибором власних уподобань.

3.3.1. PHP

PHP - це скриптова мова програмування, яку було розроблено для створення динамічних веб-сайтів. У наш час є найпоширенішою мовою для веб-серверів, яка підтримується абсолютною більшістю хост-провайдерів. Мова програмування PHP надає багато можливостей. При певному рівні володіння мовою, можна не лише створювати сценарії для веб-сторінок, а також повноцінні програми.

В даному додатку на мові програмування PHP створено повністю серверна частина системи.

4. Архітектура програмного продукту

На рисунку 3.5 зображено файлова структура проекту яка була отримана в терміналі за допомогою команди *tree*.

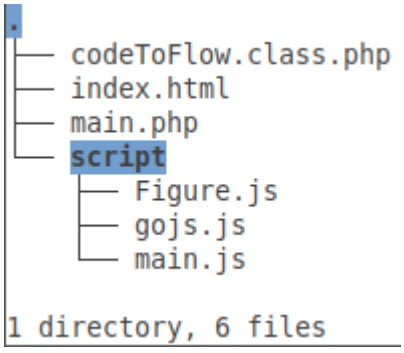


Рис. 3.5 Файлова структура проекту

Розглянемо більш детально кожен файл та каталог, та їх призначення:

index.html - головний файл проекту. Традиційно так склалось, що саме файли з назвою index є точкою входу до веб-сайту. При отриманні запиту веб-сервером за замовчуванням в першу чергу шукаються файли index.html, index.htm, index.php, index.php3, index.phtml, index.shtml. Якщо таких не знайдено, то веб-сервер може повернути файлову структуру кореневого каталогу або сформувати помилку з кодом 403 або 404, після чого браузер

буде розуміти що запитувана сторінка не знайдена.

В даному проекті в файлі *index.html* міститься HTML розмітка головної (єдиної) сторінки, яку може бачити користувач. В цій розмітці знаходяться теги для відображення поля вводу коду, кнопки для запуску трансляції коду, та контейнер для відображення готової блок-схеми, панелі для редагування блок-схеми. Також в головному файлі проекту відбувається підключення всіх JS бібліотек, необхідних для роботи системи, та підключення таблиці стилів CSS.

main.php - головна точка входу до серверної частини. Файл написан на мові програмування PHP та містить в собі встановлення налаштувань для відображення та обробки всіх помилок, що можуть виникнути в процесі роботи системи.

Також в цьому файлі підключається класс для перетворення програмного коду на масив даних для подальшого відображення блок-схем на стороні клієнта. Оскільки весь код в першу чергу надсилається до цього файлу було вирішено зробити можливим надсилати код через протокол HTTP методами POST та GET. Далі в файлі слідує створення об’єкту класса CodeToFlow, виклик методу початку конвертації, перетворення масиву результата у вигляд JSON об’єкту, надсилання відповіді клієнту.

codeToFlow.class.php - файл класса, де знаходиться алгоритм обробки коду та генерація результату у вигляді масиву даних. Оскільки PHP є об’єктно-орієнтованою мовою програмування, єдиний класс що міститься в цьому файлі побудований з використанням всіх парадигм ООП, зокрема інкапсуляції.

Єдиний метод що може бути викликано за межею класу є метод *convert()*. Саме з нього і починається процес генерації масиву даних, для відображення блок-схеми, з програмного коду.

script/ - тека, в якій містяться додаткові скрипти для роботи системи, які підключаються статично до проекту, тобто без необхідності підключення до мережі інтернет.

main.js - файл, що написано мовою програмування JavaScript, в якому сконцентровано основну логіку роботи візуальної частини системи. В цьому файлі відбувається збереження коду до наступного візиту, надсилання запиту з програмним кодом до сервера, збереження блок-схеми як зображення, завантаження коду з файлу та ініціалізація панелі з блок-схемою.

gojs.js - цей файл є ядром бібліотеки GoJS. Тут відбувається візуалізація блок-схеми, обробка масиву даних, що надсилається сервером тощо.

Figure.js - цей файл є вспоміжним файлом в роботі GoJS, в якому описуються всі можливі фігури, які представлені в цій бібліотеці, та можуть бути використані при розробці блок-схем.

Далі буде розглянута функціональність сервісу. Оскільки інтерфейс це перше що бачить користувач, та саме з графічного середовища починається обробка коду, доцільно в першу чергу описати саме клієнтську сторону системи.

На рисунку 3.6 зображено інтерфейс користувача, де можна побачити всі компоненти.

Розберемо компоненти з яких складається интерфейс:

- Текстове поле - поле куди користувач може вставити з буферу обміну код, з якого потім буде генеруватись блок-схема. Це текстове поле зроблено у вигляді текстового редактора, в якому зазвичай пишуть код. Також задля зручного використання текстового поля натискання клавіші Tab додає 4 пробіли, а не обирає наступне поле, як встановлено за замовчуванням в більшості сучасних браузерів.
- Завантаження файлу - поле вводу з типом “файл”. Зараз можливе

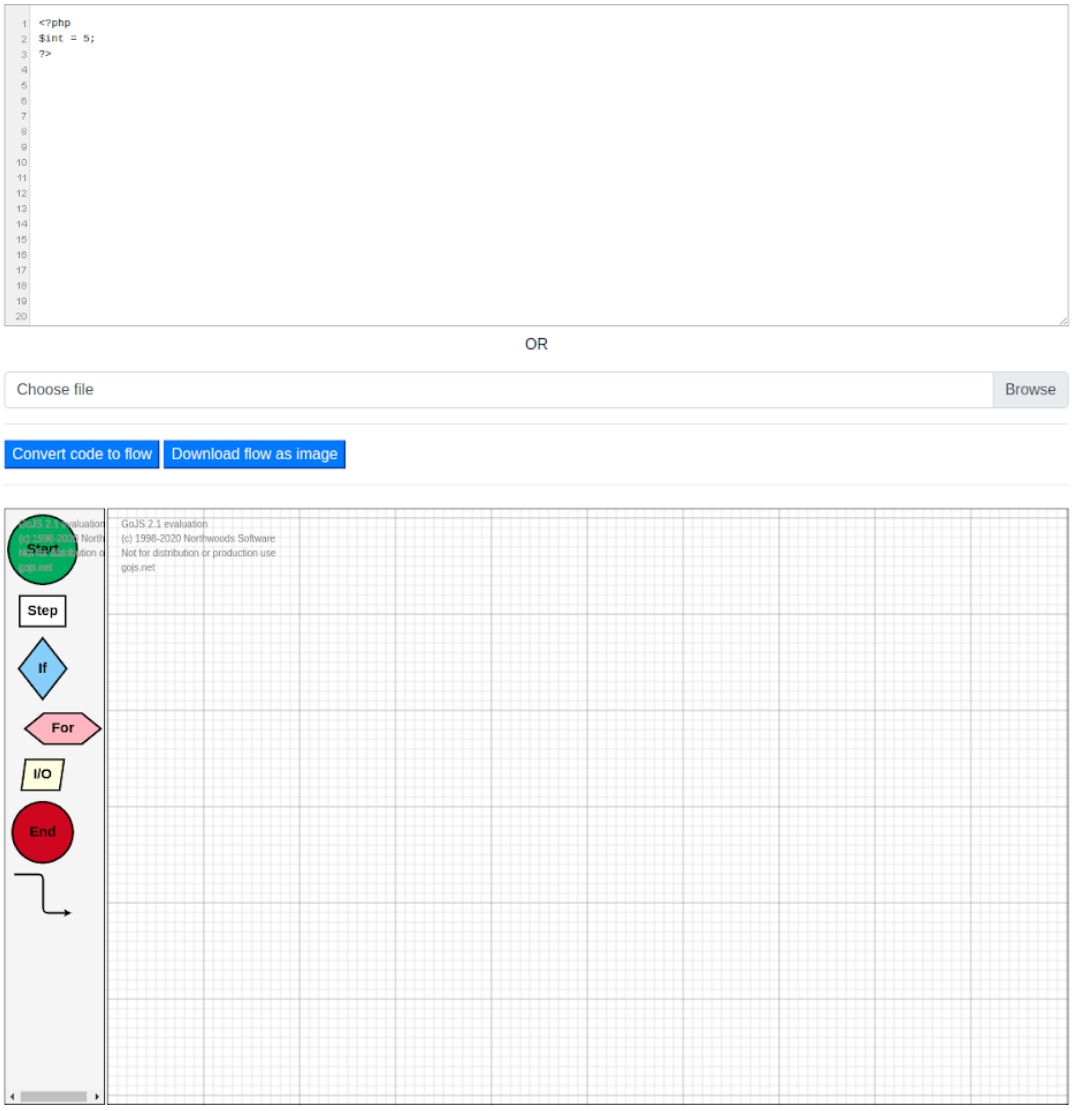


Рис 3.6

завантаження лише файлів з розширенням *.php* оскільки наразі система підтримує лише мову програмування PHP. При виборі та завантаженні файлу в систему, зміст файлу одразу вставляється в текстове поле, для подальшої обробки.

- Кнопка початку обробки - кнопка яка відправляє на сервер зміст текстового поля методом POST. Цей метод було обрано для того, щоб уникнути обмеження на розмір запиту. Також після натискання кнопки для подальшої обробки коду, він зберігається в локальному сховищі браузера користувача, щоб уникнути втрати коду в разі непередбачених

ситуацій, як наприклад раптове вимкнення комп’ютера.

Кнопка збереження блок-схеми - кнопка, що зберігає створену блок-схему на комп’ютері користувача.

Панель з відображенням блок-схем - панель, що створено бібліотекою GoJS, та в якій відображаються готові блок-схеми після отримання масиву даних з сервера. Всі елементи, що містяться в цій панелі, можна переміщувати за допомогою комп'ютерної миші; також на панелі з блок-схемою можна редагувати розмір, та текст всіх елементів.

Дві лінії - для логічного розділу частин інтерфейсу використовуються горизонтальні лінії.

Далі розглянемо послідовність роботи системи.

Користувач відкриває в браузері сторінку з веб-сайтом, на якому розташовано даний сервіс. При першому візиті в текстовому полі можна побачити простий шаблон коду на мові програмування PHP (так як наразі сервіс пропонує підтримку лише цієї мови). Користувач має можливість завантажити файл з розширенням *.php*, зміст якого потім буде завантажений до текстового поля автоматично.

```
$(document).ready(function() {

    // Load previous code from storage
    if (localStorage.getItem(LOCAL_STORAGE_NAME)) {
        document.getElementById('mySavedModel').value = localStorage.getItem(LOCAL_STORAGE_NAME);
    }

    init();
});

$(window).bind('beforeunload', function () {
    saveCode();
});

function saveCode() {
    if (!!document.getElementById('mySavedModel').value) {
        localStorage.setItem(LOCAL_STORAGE_NAME, document.getElementById('mySavedModel').value);
    } else {
        localStorage.removeItem(LOCAL_STORAGE_NAME);
    }
}
```

Рис. 3.7

Якщо клієнт вже користувався цим сайтом та вводив будь-який код, його має бути збережено в браузері та завантажено в текстове поле. На зображені (Рис. 3.7) продемонстровано частину коду яка відповідає за збереження та завантаження коду з локального сховища.

Після маніпуляцій для завантаження або написання коду, користувач має можливість дати команду початку обробки коду. При натисканні відповідної кнопки система зберігає код, введений користувачем, у локальному сховищі, після чого надсилає запит до серверу з програмним кодом. На рисунку (Рис. 3.8) зображена частина коду, в якій відбувається надсилання коду до серверу.

```
function convertCode2Flow() {
    saveCode();

    $.ajax({
        url: 'main.php',
        type: 'POST',
        data: {
            'code': document.getElementById('mySavedModel').value
        },
        success: function(data, textStatus, xhr) {
            if ('success' !== textStatus) {
                console.log(xhr);
                alert('Something goes wrong!');
                return;
            }

            let res = JSON.parse(data)

            if (res['err']) {
                console.log(res['err']);
                alert('There is an internal error');
                return;
            }

            myDiagram.model = go.Model.fromJson(res);      You, a few
            var pos = myDiagram.model.modelData.position;
            if (pos) myDiagram.initialPosition = go.Point.parse(pos);
        }
    });
}
```

Рис. 3.8

Коли сервер отримує запит в першу чергу система встановлює необхідні налаштування для реакції та показу помилок та попереджень, які можуть виникнути в процесі роботи системи (Рис 3.9).

```
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
```

Рис. 3.9

Пісня встановлення налаштувань підключається основний файл з классом для трансляції коду в масив даних, та записується отриманий сервером код у змінну для зручності (Рис 3.10).

Оскільки код може бути отриманий методами GET та POST, системи розглядає обидва методи, і в залежності від наявності змінних (даних що надійшли), система отримує програмний код, що надіслав користувач.

Пріоритет встановлений на методі POST, оскільки використання методу POST в даному випадку є більш доцільним, враховуючи головну різницю між методами. У випадку якщо код не було надіслано, сервер припиняє обробку запиту, тобто повертає порожній результат.

```
require_once 'codeToFlow.class.php';

$sCode = $_POST['code'] ?? $_GET['code'] ?? null;

if (empty($sCode)) return;
```

Рис. 3.10

Якщо код все ж таки був надісланий, та успішно записаний у змінну, відбувається створення об’єкту класу *codeToFlow* та виклик методу початку обробки (Рис. 3.11). Для того щоб система працювала безперебійно, було вирішено обрамити виклик методу *convert()* в блок для обробки помилок *try … catch.*

```
$converter = new CodeToFlow();

try {
    echo json_encode($converter->convert($sCode));
} catch (Throwable $e) {
    echo json_encode(['err' => $e->getMessage()]);
}
```

Рис. 3.11

Переведення результату до вигляну JSON, відбувається одразу, без зайвих змінних. Це також дає невелике прискорення на обробку. У випадку, якщо система перерветься через виникнення помилки, блок *try … catch* перехопить помилку, та поверне користувачу масив, з елементом *‘err’*, в якому міститься інформація, яку надає помилка.

Після виклику методу *convert()* вихідний код розбивається на токени. Токени - найменша структурна одиниця коду. Вони відіграють ключову роль у роботі компілятора чи інтерпретатора коду.

Саме завдяки токенам комп’ютер відрізняє змінну від тексту, функцію від команди завершення роботи, тощо. Оскільки обробка коду відбувається мовою PHP, та підтримка мови програмування PHP є наразі єдиною для системи, то використовуючи функцію *token_get_all()* можна отримати масив з токенами замість вихідного коду (Рис. 3.12).

```
$arTokens = token_get_all($sCode);

$this->arResult = [
    'class' => 'GraphLinksModel',
    'nodeDataArray' => [],
    'linkDataArray' => [],
];
```

Рис. 3.12

Також на початку функції відбувається чорнове оформлення результату (Рис. 3.12). Формується масив, де заготовлено наступні ключі:

- Клас діаграми
- Масив для даних щодо елементів
- Масив для даних щодо зв'язків між елементами.

Після розбиття коду на токени відбувається обробка токенів в циклі, де кожний токен обробляється згідно зі своїми властивостями.

Далі розглянемо деякі фрагменти коду, які мають покращувати клієнт-серверну систему трансляції програмного коду, що розробляється.

Однією з доволі значних переваг є завантаження файлу та відображення його змісту без завантаження на сервер. Це стає можливим за допомогою об’єкту класса *FileReader*, який дозволяє асинхронно читати зміст файлів, що знаходяться на комп’ютері користувача [11] (Рис. 3.13). Коли користувач натискає на кнопку завантаження, браузер пропонує обрати файл, які мають бути завантажені, після чого реагує тригер *change*, і потім відбувається читання файлу.

Якщо файл вже був завантажений попередньо, та користувач завантажує інший, то система перезаписує код в текстовому полі на новий. Також для поля з завантаженням файлу встановлене налаштування, що за один раз можливо завантажити лише один файл.

Із збільшенням кількості мов програмування, що підтримує система, буде збільшена кількість розширень файлів, що дозволяється завантажувати до системи.

```
$('#loadFile').bind('change', function (e) {
    if(document.querySelector("#loadFile").files.length == 0) {
        alert('No file selected!');
        return;
    }

    var file = document.querySelector("#loadFile").files[0];
    var reader = new FileReader();

    reader.addEventListener('loadstart', function() {
        console.log('File reading started');
    });
    reader.addEventListener('load', function(e) {
        var text = e.target.result;
        document.getElementById('mySavedModel').value = text;
    });
    reader.addEventListener('error', function() {
        alert('Error : Failed to read file');
    });

    reader.readAsText(file);
});
```

Рис. 3.13

Також хочу звернути увагу на можливість використання клавіші *Tab*, для вставки відступу в текстовому редакторі.

Це відбувається наступним чином: для текстового поля встановлена обробка події при натисканні будь-якої клавіші. Далі відбувається перевірка коду клавіші (у клавіші *Tab* код 9), і якщо коди співпадають, то примусово припиняється стандартна дія клавіші, і в місці де стоїть курсор вставляється 4 символи пробілу (Рис. 3.14).

```
$('#mySavedModel').keydown(function (e) {
    var keyCode = e.keyCode || e.which;

    if (keyCode === 9) {
        e.preventDefault();
        var start = this.selectionStart;
        var end = this.selectionEnd;

        spaces = "    ";
        this.value = this.value.substring(0, start) + spaces + this.value.substring(end);

        this.selectionStart = this.selectionEnd = start + spaces.length;
    }
});
```

Рис. 3.14

Однією з цікавих особливостей функціоналу є можливість збереження блок-схеми як зображення.

Під час реалізації виникла проблема, пов’язана з тим, що блок-схема відображається за допомогою технології canvas. Для збереження блок-схеми необхідно спочатку отримати текстову репрезентацію зображення, змінити тип даних з *image/png* на *image/octet-stream*, і потім встановити отриманий результат як поточну сторінку (Рис. 3.15).

```
function saveFlowAsImage() {
    var canvas = document.getElementsByTagName('canvas')[1];
    var image = canvas.toDataURL("image/png").replace("image/png", "image/octet-stream");
    window.location.href = image;
}
```

Рис. 3.15

Браузер спробує перейти за посиланням, оскільки це не веб-сторінка,

а текстове представлення зображення, браузер автоматично збереже файл на комп’ютері користувача.

5. Демонстрація роботи розробленої системи

На зображеннях (Рис. 3.16-19) продемонстровано як відбувається робота прототипу системи. На даному етапі реалізована підтримка більшості типів алгоритмів. Усі зображення були отримані за допомогою вбудованого збереження зображень, що реалізоване в системі.

В прикладах представлені алгоритми, які були використані в зображеннях (Рис. 2.12-14), а також деякі відомі алгоритми.

5.1. Покроковий алгоритм

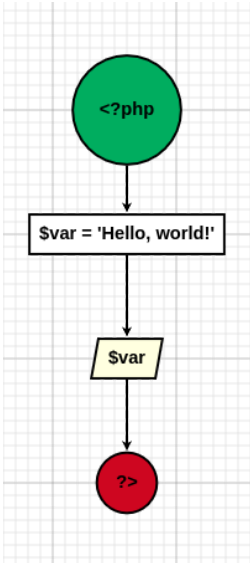


Рис. 3.16

5.2. Алгоритм з умовою

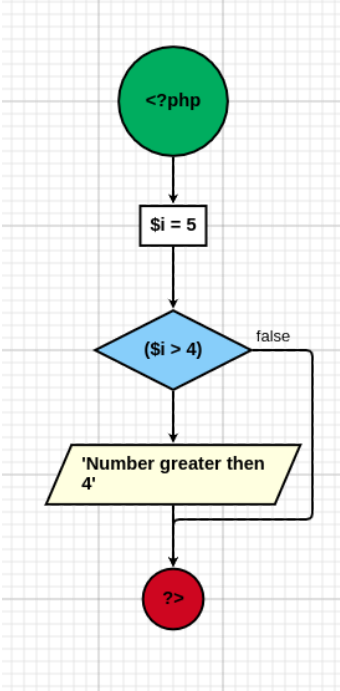


Рис 3.17

5.3. Алгоритм з циклом

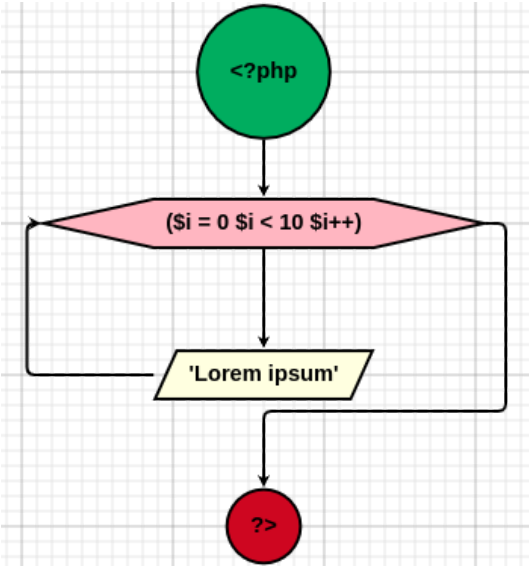


Рис 3.18

5.4. Bubble sort

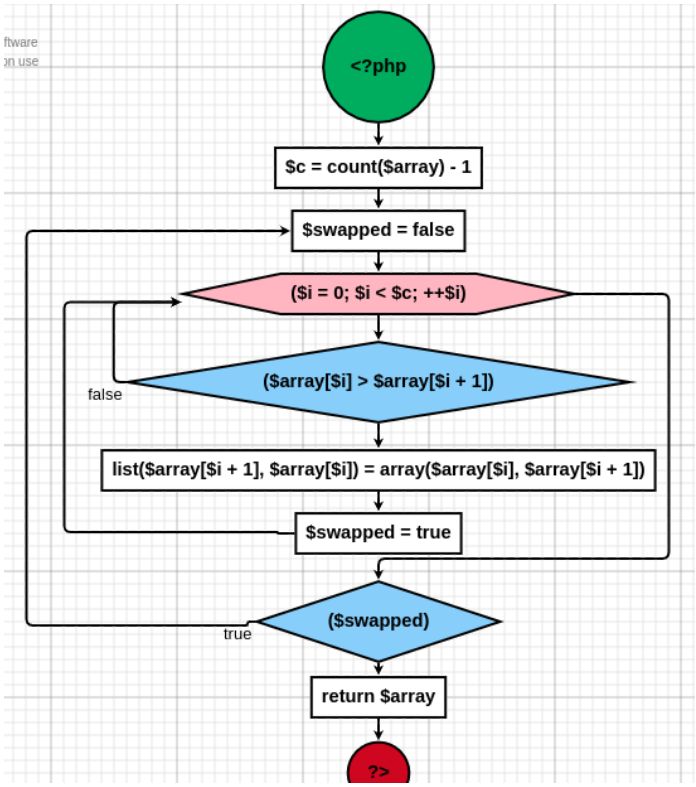


Рис 3.19

ВИСНОВОК РОЗДІЛУ 3

В фінальному розділі даної роботи було детально описано процес створення клієнт-серверної системи трансляції програмного коду між графічним та текстовим представленням. Прототип системи було багаторазово протестовано за допомогою великої кількості кодів.

Наразі версія прототипу програмного продукту є стабільною, але слід зазначити, що випуск повноцінної програми зі взаємною трансформацією алгоритмів із текстового у візуальне представлення блок-схем потребує значної доробки та додаткового тестування. Також реалізовано підтримку таких блоків:

- Процес;
- Дані;
- Підготовка;
- Умова;
- Лінія.

За допомогою вищеперелічених блоків було реалізовано побудову наступних конструкцій:

- Послідовний алгоритм;
- Алгоритми з умовою;
- Алгоритми з циклом;
- Алгоритми з післяумовою (do … while).

На даний момент, наявність лише цих конструкцій та блоків є достатньою для побудови більшості алгоритмів.

Більшість недоліків розробленого прототипу вже є скоригованою у порівнянні з аналогічними продуктами. Також враховано переваги.

Отже, у результаті створення повноцінної версії даної клієнт-серверної системи, всі проаналізовані програмні продукти може бути замінено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DAXX. How Many Software Developers Are in the US and the World? [Електронний ресурс] // DAXX. – 2020. – Режим доступу до ресурсу: https://www.daxx.com/blog/development-trends/number-software-developers-world.
2. ASEE/PSW-2015 Conference Proceedings. // American Society for Engineering Education. – 2015. – С. 158.
3. ГОСТ 19.701-90. // Единая система программной документации: Сб. ГОСТов. / , 2005. – (Стандартинформ). – С. 5–22.
4. Developer Survey Results [Електронний ресурс] // Stack Overflow. – 2019. – Режим доступу до ресурсу: https://insights.stackoverflow.com/survey/2019/#technology- -platforms.
5. Front end та back end [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Front_end_%D1%82%D0%B0_back_end.
6. Overview. // jQuery. Web application library / , 2015. – С. 1.
7. Н. М. Васильків, Л. О. Васильків. Опорний конспект лекцій з дисципліни «Основи алгоритмізації» спеціальність «Комп'ютерні системи та мережі». — Тернопіль : Економічна думка, 2005. — 32 с.
8. Алгоритм [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC.
9. Hypertext Transfer Protocol -- HTTP/1.1. // Hypertext Transfer Protocol -- HTTP/1.1 / , 1999. – (The Internet Society). – С. 7.

10. WWW FAQs: What is the maximum length of a URL? [Електронний ресурс] // Boutell. – 2006. – Режим доступу до ресурсу: https://web.archive.org/web/20170503192739/https://boutell.com/newfaq/misc/urllength.html.
11. FileReader [Електронний ресурс] // Mozilla. – 2019. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/FileReader.

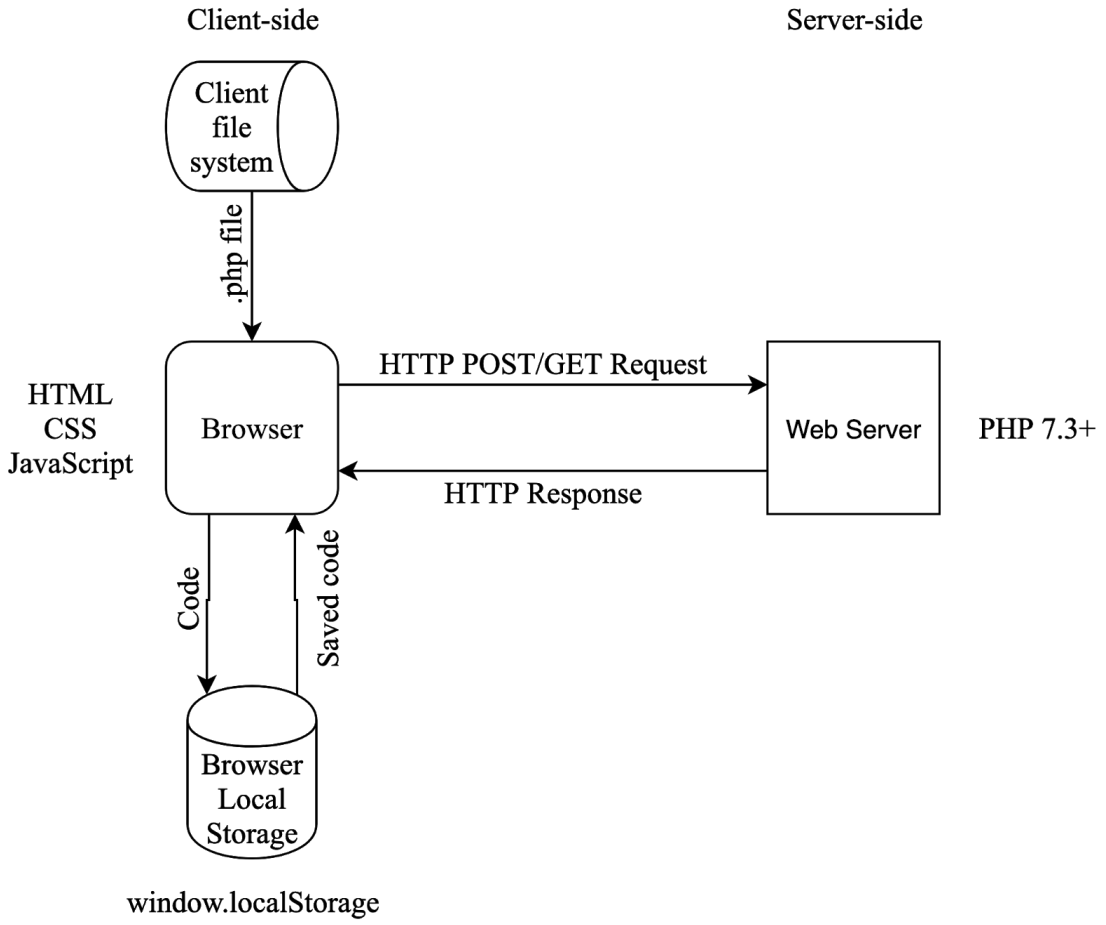
Додатки

Додаток А

Структурна схема. Зв’язок між вузлами

Аркушів 1

Київ 2020



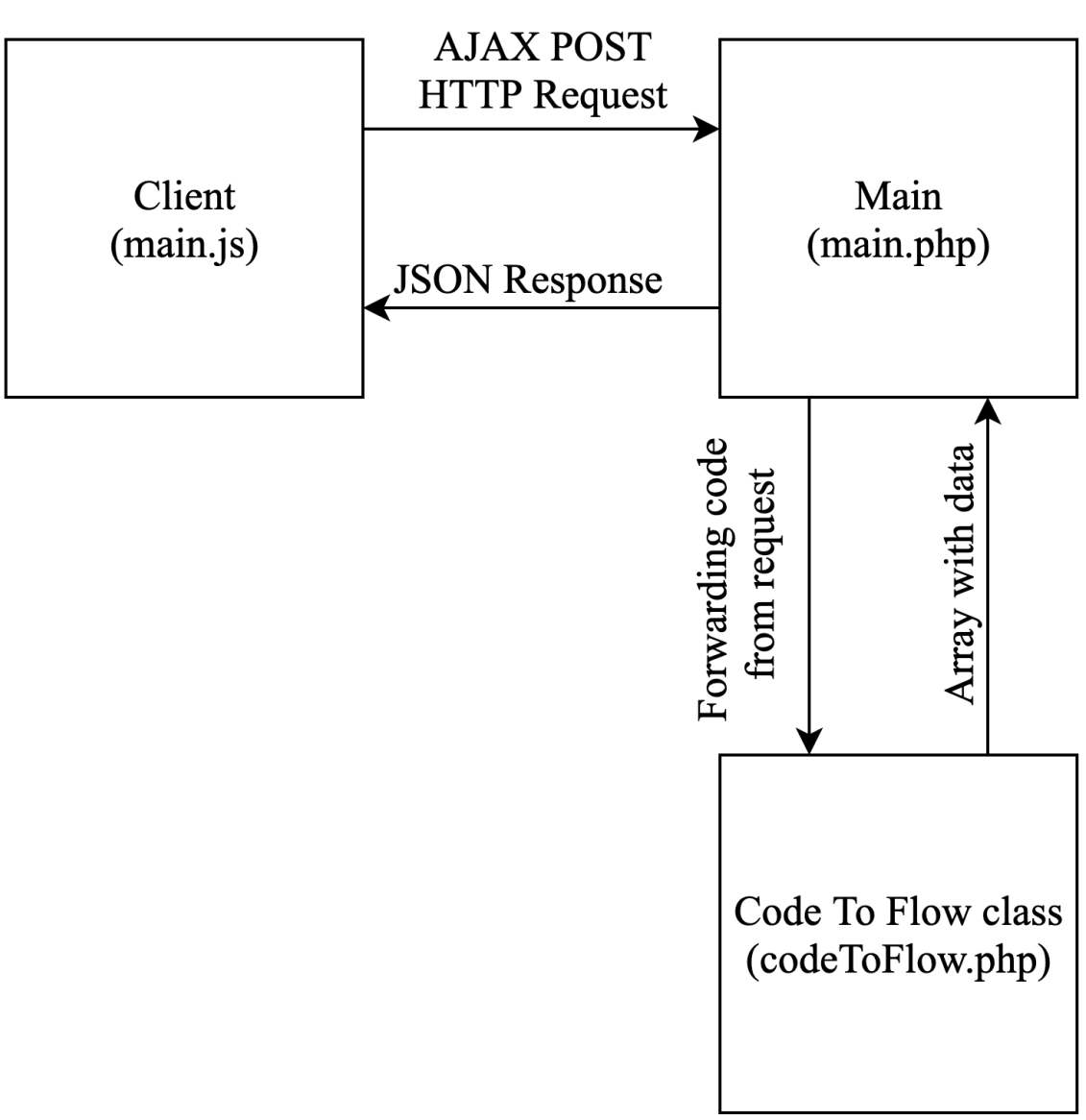
					ІАЛЦ.467100.004 Д1		
Змн	Літ.	№ докум.	Підпис	Дат			
Розроб.		Замалдінов О. О.			Структурна схема Зв’язок між вузлами	Літ.	Арк. 1 / Аркушів 1
Перевір.		Алещенко О. В.					
Т. Контр						НТУУ “КПІ”, ФІОТ, гр. ІО-62	
Н. Контр		Сімоненко В. П.					
Затверд.		Стіренко С. Г.					

Додаток В

Функціональна схема. Взаємодія модулів програми

Аркушів 1

Київ 2020



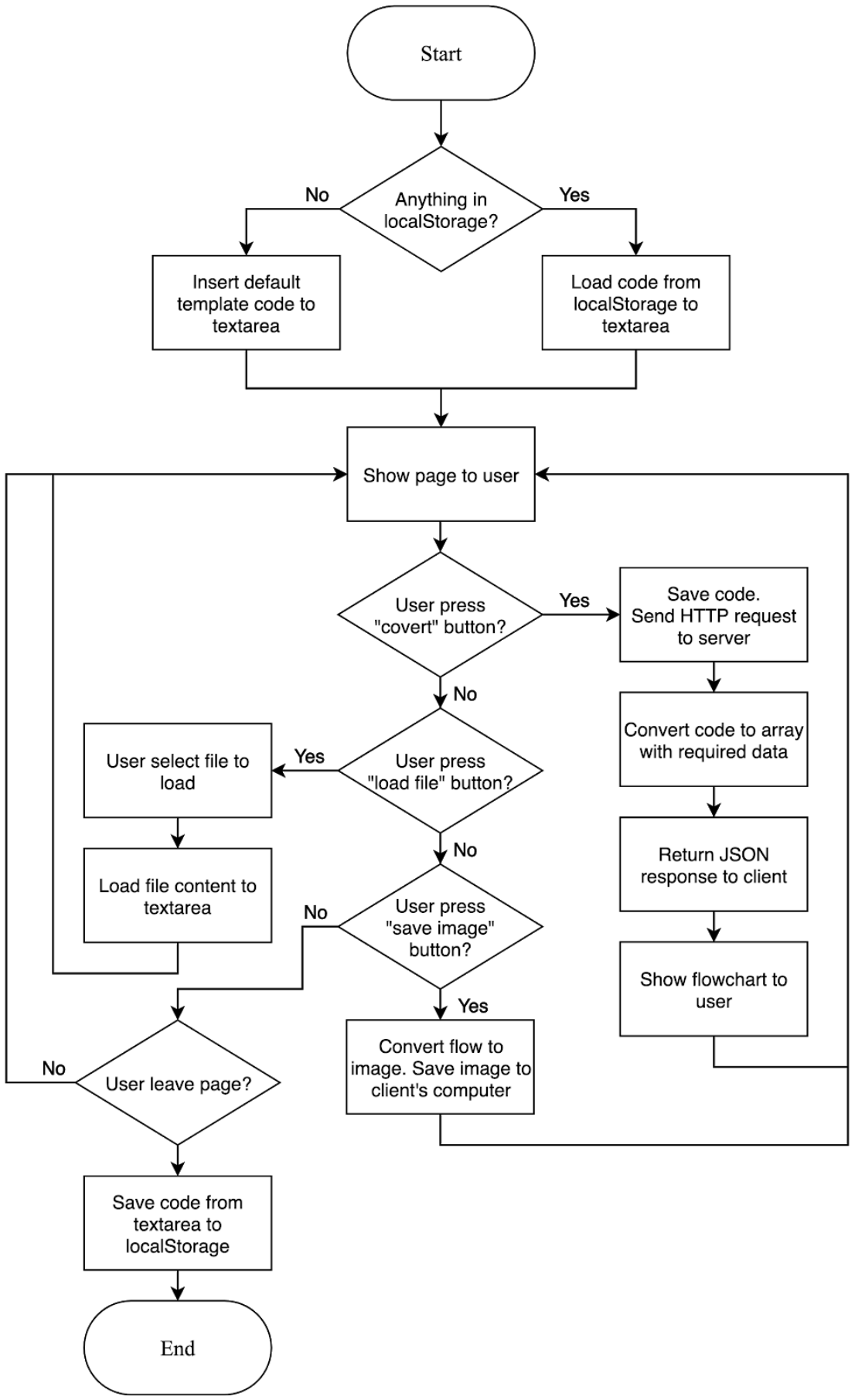
					ІАЛЦ.467100.005 Д2		
Змн	Літ.	№ докум.	Підпис	Дат			
Розроб.		Замалдінов О. О.			Функціональна схема Взаємодія модулів програми	Літ.	Арк. 1 / Аркушів 1
Перевір.		Алещенко О. В.					
Т. Контр						НТУУ “КПІ”, ФІОТ, гр. ІО-62	
Н. Контр		Сімоненко В. П.					
Затверд.		Стіренко С. Г.					

Додаток С

Принципова схема. Алгоритм обробки запитів

Аркушів 1

Київ 2020



					ІАЛЦ.467100.006 ДЗ		
Змн	Літ.	№ докум.	Підпис	Дат			
Розроб.		Замалдінов О. О.			Принципова схема. Алгоритм обробки запитів	Літ.	Арк. 1 / Аркушів 1
Перевір.		Алещенко О. В.					
Т. Контр						НТУУ “КПІ”, ФІОТ, гр. ІО-62	
Н. Контр		Сімоненко В. П.					
Затверд.		Стіренко С. Г.					

Додаток D

Ключові елементи коду програми

Аркушів 11

Київ 2020

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
   <title>Oleksii Zamaldinov</title>
   <style>
       #mySavedModel {
           background: url(http://i.imgur.com/2cOaJ.png);
           background-attachment: local;
           background-repeat: no-repeat;
           padding-left: 35px;
           padding-top: 11px;
           font-size: 11px;
       }
   </style>
</head>
<body>
   <div class="content container my-4">
       <div class="row">
           <div class="col-12">
               <textarea class="w-100 text-monospace" id="mySavedModel" rows="20"><?php
$int = 5;
?></textarea>
               <div class="text-center">
                   <p>OR</p>
               </div>
               <div class="custom-file">
                   <input type="file" class="custom-file-input" id="loadFile" accept=".php">
                   <label class="custom-file-label" for="loadFile">Choose file</label>
               </div>
               <hr>
               <button class="btn-primary" onclick="convertCode2Flow()">Convert code to
flow</button>
               <!-- <button class="btn-primary" onclick="convertFlow2Code()">Convert flow to
code</button> -->
               <button class="btn-primary" onclick="saveFlowAsImage()">Download flow as
image</button>
           </div>
       </div>
       <hr>
       <div class="row my-4">
           <div class="col-12" style="width: 100%; display: flex; justify-content:
space-between">
               <div id="myPaletteDiv" style="width: 105px; margin-right: 2px; background-color:
whitesmoke; border: solid 1px black"></div>
               <div id="myDiagramDiv" style="flex-grow: 1; height: 620px; border: solid 1px
black"></div>
           </div>
       </div>
```

```
    </div>


    <!-- JavaScript -->
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"
integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0=" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
    <script src="script/gojs.js"></script>
    <script src="script/Figure.js"></script>
    <script src="script/main.js"></script>
</body>
</html>
```

main.js

```
let LOCAL_STORAGE_NAME = 'code';

$(document).ready(function() {

    // Load previous code from storage
    if (localStorage.getItem(LOCAL_STORAGE_NAME)) {
        document.getElementById('mySavedModel').value = localStorage.getItem(LOCAL_STORAGE_NAME);
    }

    init();
});

$(window).bind('beforeunload', function () {
    saveCode();
});

function saveCode() {
    if (!!document.getElementById('mySavedModel').value) {
        localStorage.setItem(LOCAL_STORAGE_NAME, document.getElementById('mySavedModel').value);
    } else {
        localStorage.removeItem(LOCAL_STORAGE_NAME);
    }
}

$('#loadFile').bind('change', function (e) {
    if(document.querySelector("#loadFile").files.length == 0) {
        alert('No file selected!');
        return;
    }

    var file = document.querySelector("#loadFile").files[0];
    var reader = new FileReader();

    reader.addEventListener('loadstart', function() {
        console.log('File reading started');
    });
```

```
    reader.addEventListener('load', function(e) {
        var text = e.target.result;
        document.getElementById('mySavedModel').value = text;
    });
    reader.addEventListener('error', function() {
        alert('Error : Failed to read file');
    });

    reader.readAsText(file);
});

$('#mySavedModel').keydown(function (e) {
    var keyCode = e.keyCode || e.which;

    if (keyCode === 9) {
        e.preventDefault();
        var start = this.selectionStart;
        var end = this.selectionEnd;

        spaces = "    ";
        this.value = this.value.substring(0, start) + spaces + this.value.substring(end);

        this.selectionStart = this.selectionEnd = start + spaces.length;
    }
});

function convertCode2Flow() {
    saveCode();

    $.ajax({
        url: 'main.php',
        type: 'POST',
        data: {
            'code': document.getElementById('mySavedModel').value
        },
        success: function(data, textStatus, xhr) {
            if ('success' !== textStatus) {
                console.log(xhr);
                alert('Something goes wrong!');
                return;
            }

            let res = JSON.parse(data)

            if (res['err']) {
                console.log(res['err']);
                alert('There is an internal error');
                return;
            }

            myDiagram.model = go.Model.fromJson(res);
            var pos = myDiagram.model.modelData.position;
            if (pos) myDiagram.initialPosition = go.Point.parse(pos);
        }
    });
}
```

```
function saveFlowAsImage() {
   var canvas = document.getElementsByTagName('canvas')[1];
   var image = canvas.toDataURL("image/png").replace("image/png", "image/octet-stream");
   window.location.href = image;
}
```

main.php

```
<?php

ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);

require_once 'codeToFlow.class.php';

$sCode = $_POST['code'] ?? $_GET['code'] ?? null;

if (empty($sCode)) return;

$converter = new CodeToFlow();

try {
   echo json_encode($converter->convert($sCode));
} catch (Throwable $e) {
   echo json_encode(['err' => $e->getMessage()]);
}
```

codeToFlow.class.php

```
<?php

class CodeToFlow {
   private const FIG_START = 1;
   private const FIG_END = 2;
   private const FIG_IO = 3;
   private const FIG_IF = 4;
   private const FIG_FOR = 5;
   private const FIG_DO = 6;
   private const FIG_WHILE = 7;

   private const BLOCK_START = 1;
   private const BLOCK_TYPE = 2;
   private const BLOCK_NO_BRACKET = 3;

   private const LINK_FLAG_FROM_LEFT = 1;
   private const LINK_FLAG_FROM_RIGHT = 2;
   private const LINK_FLAG_FROM_TOP = 4;
   private const LINK_FLAG_TO_LEFT = 8;
   private const LINK_FLAG_TO_RIGHT = 16;
   private const LINK_FLAG_TO_BOTTOM = 32;

   private const PEND_START = 1;
   private const PEND_TEXT = 2;
```

```
    private const PEND_END = 3;
    private const PEND_FLAGS = 4;

    private const AR_PARAMS = [
        self::FIG_START => [
            'figure' => 'Circle',
            'fill' => '#00AD5F',
        ],
        self::FIG_END => [
            'figure' => 'Circle',
            'fill' => '#CE0620',
        ],
        self::FIG_IO => [
            'figure' => 'Input',
            'fill' => 'lightyellow',
        ],
        self::FIG_IF => [
            'figure' => 'Diamond',
            'fill' => 'lightskyblue',
        ],
        self::FIG_FOR => [
            'figure' => 'Hexagon',
            'fill' => 'lightpink',
            'angle' => '90',
        ],
        self::FIG_WHILE => [
            'figure' => 'Diamond',
            'fill' => 'lightskyblue',
        ],
    ];

    private $iTokenNumber = -1;
    private $sPart = '';
    private $arResult = [];
    private $sLinkText = '';
    private $bAddNextLink = true;
    private $arPendingLink = [];
    private $bIsContinueWas = false;
    private $iType = 0;

    public function convert($sCode): array {
        $arTokens = token_get_all($sCode);

        $this->arResult = [
            'class' => 'GraphLinksModel',
            'nodeDataArray' => [],
            'linkDataArray' => [],
        ];

        $arBlocks = [];

        foreach ($arTokens as $token) {
            if (is_string($token)) {
                if (';' === $token) {
                    if (self::FIG_IO === $this->iType) {
                        $this->addNode();
                    } elseif (self::FIG_WHILE === $this->iType) {
```

```
                    $this->addNode();
                    continue;
                } elseif (0 === $this->iType) {
                    foreach (array_reverse($arBlocks) as $arBlock) {
                        if ($arBlocks[count($arBlocks) - 1][self::BLOCK_NO_BRACKET] ?? false)
{
                            $arBlockStart = array_pop($arBlocks) ?? null;
                            if (empty($arBlockStart)) continue;
                            if (self::FIG_IF === $arBlockStart[self::BLOCK_TYPE]) {
                                $this->arPendingLink = [
                                    self::PEND_START => $arBlockStart[self::BLOCK_START],
                                    self::PEND_TEXT => 'false',
                                ];
                                $this->addNode();
                            }
                            if (self::FIG_FOR === $arBlockStart[self::BLOCK_TYPE]) {
                                $this->addLink($this->iTokenNumber + 1,
$arBlockStart[self::BLOCK_START], null, self::LINK_FLAG_FROM_LEFT | self::LINK_FLAG_TO_BOTTOM);
                                $this->addLink($arBlockStart[self::BLOCK_START],
$this->iTokenNumber, null, self::LINK_FLAG_FROM_TOP);
                                $this->addNode(null, null);
                                $this->bAddNextLink = false;
                            }
                            $this->addNode();
                        } else {
                            break;
                        }
                    }
                    $this->addNode();
                    continue;
                }
            } elseif ('{' === $token) {
                if (self::FIG_IF === $this->iType || self::FIG_FOR === $this->iType)
$arBlocks[count($arBlocks) - 1][self::BLOCK_NO_BRACKET] = false;
                if (self::FIG_IF === $this->iType) $this->sLinkText = 'true';
                $this->addNode();
                continue;
            } elseif ('}' === $token) {
                if (self::FIG_DO === $arBlocks[count($arBlocks) - 1][self::BLOCK_TYPE])
continue;
                $arBlockStart = array_pop($arBlocks) ?? null;
                if (empty($arBlockStart)) continue;
                if (self::FIG_IF === $arBlockStart[self::BLOCK_TYPE]) {
                    $this->arPendingLink = [
                        self::PEND_START => $arBlockStart[self::BLOCK_START],
                        self::PEND_TEXT => 'false',
                    ];
                    $this->addNode();
                }
                if (self::FIG_FOR === $arBlockStart[self::BLOCK_TYPE]) {
                    $this->addLink($this->iTokenNumber + 1, $arBlockStart[self::BLOCK_START],
null, self::LINK_FLAG_FROM_LEFT | self::LINK_FLAG_TO_BOTTOM);
                    $this->addLink($arBlockStart[self::BLOCK_START], $this->iTokenNumber,
null, self::LINK_FLAG_FROM_TOP);
                    $this->addNode(null, null);
                    $this->bAddNextLink = false;
                }
```

```
                    $this->addNode();
                    continue;
                }
                $this->sPart .= $token;
                continue;
            }
            switch ($token[0]) {
                case T_OPEN_TAG:
                case T_CLOSE_TAG:
                    if (!empty($this->sPart)) $this->addNode();
                    $this->iType = T_OPEN_TAG === $token[0] ? self::FIG_START : self::FIG_END;
                    $this->addNode($token[1]);
                    break;
                case T_ECHO:
                    $this->addNode();
                    $this->iType = self::FIG_IO;
                    break;
                case T_IF:
                    $this->addNode();
                    $arBlocks[] = [
                        self::BLOCK_START => $this->iTokenNumber,
                        self::BLOCK_TYPE => self::FIG_IF,
                        self::BLOCK_NO_BRACKET => true,
                    ];
                    $this->iType = self::FIG_IF;
                    break;
                case T_FOR:
                case T_FOREACH:
                    $this->addNode();
                    $arBlocks[] = [
                        self::BLOCK_START => $this->iTokenNumber,
                        self::BLOCK_TYPE => self::FIG_FOR,
                        self::BLOCK_NO_BRACKET => true,
                    ];
                    $this->iType = self::FIG_FOR;
                    break;
                case T_CONTINUE:
                    foreach (array_reverse($arBlocks) as $arBlock) {
                        if (self::FIG_FOR === $arBlock[self::BLOCK_TYPE]) {
                            $this->addNode();
                            $this->addLink($this->iTokenNumber + 1, $arBlock[self::BLOCK_START],
null, self::LINK_FLAG_FROM_LEFT | self::LINK_FLAG_TO_LEFT);
                            $this->bIsContinueWas = true;
                            break;
                        }
                    }
                    break;
                case T_DO:
                    $this->addNode();
                    $arBlocks[] = [
                        self::BLOCK_START => $this->iTokenNumber,
                        self::BLOCK_TYPE => self::FIG_DO,
                    ];
                    break;
                case T_WHILE:
                    $this->addNode();
                    $this->iType = self::FIG_WHILE;
```

```
                    if (!empty($arBlocks) && self::FIG_DO === $arBlocks[count($arBlocks) -
1][self::BLOCK_TYPE]) {
                        $arBlockStart = array_pop($arBlocks) ?? null;
                        if (empty($arBlockStart)) break;
                        $this->arPendingLink = [
                            self::PEND_END => $arBlockStart[self::BLOCK_START],
                            self::PEND_START => $this->iTokenNumber,
                            self::PEND_TEXT => 'true',
                            self::PEND_FLAGS => self::LINK_FLAG_FROM_LEFT |
self::LINK_FLAG_TO_LEFT,
                        ];
                        $this->addNode();
                    }
                    break;
                default:
                    $this->sPart .= $token[1];
                    break;
            }
        }

        if (!empty($this->sPart)) $this->addNode();

        return $this->arResult;
    }

    private function addNode(?string $sText = null) {
        $sText = trim($sText ?? $this->sPart);
        if (empty($sText)) return;
        if (';' === $sText) {
            $this->sPart = '';
            return;
        }
        $arTmp = [
            'text' => $sText,
            'key' => $this->iTokenNumber,
            'loc' => '60 '.(100 * -1 * $this->iTokenNumber),
        ];
        if (!empty($this->iType)) {
            $arTmp = array_merge(self::AR_PARAMS[$this->iType], $arTmp);
            $this->iType = 0;
        }
        $this->arResult['nodeDataArray'][] = $arTmp;
        if (-1 !== $this->iTokenNumber && $this->bAddNextLink) $this->addLink($this->iTokenNumber
+ 1, $this->iTokenNumber);
        $this->sPart = '';
        $this->iTokenNumber--;
        $this->bAddNextLink = true;
    }

    private function addLink(int $iFrom, int $iTo, ?string $sText = null, int $iFlags = 0, bool
$bIgnore = false) {
        if (!empty($this->arPendingLink)) {
            $arPendingLink = $this->arPendingLink;
            $this->arPendingLink = [];
            $this->addLink($arPendingLink[self::PEND_START] ?? $iFrom,
$arPendingLink[self::PEND_END] ?? $iTo, $arPendingLink[self::PEND_TEXT] ?? $sText,
$arPendingLink[self::PEND_FLAGS] ?? $iFlags, true);
```

```
        }
        if ($this->bIsContinueWas && !$bIgnore) {
            $this->bIsContinueWas = false;
            return;
        }
        $arTmp = [
            'from' => $iFrom,
            'to' => $iTo,
        ];
        if (!empty($sText) || !empty($this->sLinkText)) {
            $arTmp['text'] = !empty($sText) ? $sText : $this->sLinkText;
            if ('false' === $arTmp['text']) $arTmp['fromSpot'] = 'RightSide';
            $this->sLinkText = '';
        }
        if ($iFlags & self::LINK_FLAG_FROM_LEFT) $arTmp['fromSpot'] = 'LeftSide';
        if ($iFlags & self::LINK_FLAG_FROM_RIGHT) $arTmp['fromSpot'] = 'RightSide';
        if ($iFlags & self::LINK_FLAG_FROM_TOP) $arTmp['fromSpot'] = 'TopSide';
        if ($iFlags & self::LINK_FLAG_TO_LEFT) $arTmp['toSpot'] = 'LeftSide';
        if ($iFlags & self::LINK_FLAG_TO_RIGHT) $arTmp['toSpot'] = 'RightSide';
        if ($iFlags & self::LINK_FLAG_TO_BOTTOM) $arTmp['toSpot'] = 'BottomSide';
        $this->arResult['linkDataArray'][] = $arTmp;
    }
}
```